SIAM J. SCI. COMPUT.
Vol. 33, No. 4, pp. 1996–2023

HYPERGRAPH PARTITIONING-BASED FILL-REDUCING ORDERING FOR SYMMETRIC MATRICES*

ÜMIT V. ÇATALYÜREK†, CEVDET AYKANAT‡, AND ENVER KAYAASLAN‡

Abstract. A typical first step of a direct solver for the linear system $Mx = b$ is reordering of the symmetric matrix M to improve execution time and space requirements of the solution process. In this work, we propose a novel nested-dissection-based ordering approach that utilizes hypergraph partitioning. Our approach is based on the formulation of graph partitioning by vertex separator (GPVS) problem as a hypergraph partitioning problem. This new formulation is immune to deficiency of GPVS in a multilevel framework and hence enables better orderings. In matrix terms, our method relies on the existence of a *structural* factorization of the input M matrix in the form of $M = AA^T$ (or $M = AD^2A^T$). We show that the partitioning of the row-net hypergraph representation of the rectangular matrix A induces a GPVS of the standard graph representation of matrix M. In the absence of such factorization, we also propose simple, yet effective structural factorization techniques that are based on finding an edge clique cover of the standard graph representation of matrix M, and hence applicable to any arbitrary symmetric matrix M. Our experimental evaluation has shown that the proposed method achieves better ordering in comparison to state-of-the-art graph-based ordering tools even for symmetric matrices where structural $M = AA^T$ factorization is not provided as an input. For matrices coming from linear programming problems, our method enables even faster and better orderings.

Key words. fill-reducing ordering, hypergraph partitioning, combinatorial scientific computing

AMS subject classifications. 05C65, 05C85, 68R10, 68W05

DOI. 10.1137/090757575

1. Introduction.

The focus of this work is the solution of symmetric linear systems of equations through direct methods such as LU and Cholesky factorizations. A typical first step of a direct method is a heuristic reordering of the rows and columns of M to reduce *fill* in the triangular factor matrices. The fill is the set of zero entries in M that become nonzero in the triangular factor matrices. Another goal in reordering is to reduce the number of floating-point operations required to perform the triangular factorization, also known as *operation count*. It is equal to the sum of the squares of the number nonzeros of each eliminated row/column; hence it is directly related with the number of fills.

For a symmetric matrix, the evolution of the nonzero structure during the factorization can easily be described in terms of its graph representation [50]. In graph terms, the elimination of a vertex (which corresponds to a row/column of the matrix) creates an edge for each pair of its adjacent vertices. In other words, elimination of a vertex makes its adjacent vertices into a clique of size equal to its degree. In this process, the extra edges, which are added to construct such cliques, directly correspond to the fill in the matrix. Obviously, the amount of fill and operation count depends on

*Submitted to the journal's Methods and Algorithms for Scientific Computing section April 30, 2009; accepted for publication (in revised form) May 11, 2011; published electronically August 18, 2011.

http://www.siam.org/journals/sisc/33-4/75757.html

†Department of Biomedical Informatics, The Ohio State University, Columbus, OH 43210 (catalyurek.1@osu.edu). The first author's work was partially supported by U.S. DOE SciDAC Institute grant DE-FC02-06ER2775 and U.S. National Science Foundation under grants CNS-0643969, OCI-0904809, and OCI-0904802.

‡Computer Engineering Department, Bilkent University, Ankara, Turkey (aykanat@cs.bilkent.edu.tr, enver@cs.bilkent.edu.tr). The second author's work was partially supported by The Scientific and Technical Research Council of Turkey (TÜBİTAK) under project EEEAG-109E019.

the row/column elimination order. The aim of ordering is to reduce these quantities, which leads to both faster and less memory intensive solution of the linear system. Unfortunately this problem is known to be NP-hard [54]; hence we consider heuristic ordering methods.

Heuristic methods for fill-reducing ordering can be divided into mainly two categories: *bottom-up* (also called *local*) and *top-down* (also called *global*) approaches [49]. In the bottom-up category, one of the most popular ordering methods is the *minimum degree* (MD) heuristic [52] in which at every elimination step a vertex with the minimum degree, hence the name, is chosen for elimination. Success of the MD heuristic is followed by many variants of it, such as quotient minimum degree [29], multiple minimum degree (MMD) [48], approximate minimum degree (AMD) [2], and approximate minimum fill [51]. Among the top-down approaches, the most famous and influential one is surely *nested dissection* (ND) [30]. The main idea of ND is as follows. Consider a partitioning of vertices into three sets, $\mathcal{V}_1$, $\mathcal{V}_2$, and $\mathcal{V}_S$, such that the removal of $\mathcal{V}_S$, called *separator*, decouples $\mathcal{V}_1$ and $\mathcal{V}_2$. If we order the vertices of $\mathcal{V}_S$ after the vertices of $\mathcal{V}_1$ and $\mathcal{V}_2$, certainly no fill can occur between the vertices of $\mathcal{V}_1$ and $\mathcal{V}_2$. Furthermore, the elimination processes in $\mathcal{V}_1$ and $\mathcal{V}_2$ are independent tasks, and their elimination only incurs fill to themselves and $\mathcal{V}_S$. Hence, the ordering of the vertices of $\mathcal{V}_1$ and $\mathcal{V}_2$ can be computed by applying the algorithm recursively. In ND, since the quality of the ordering depends on the size of $\mathcal{V}_S$, finding a small separator is desirable.

Although the ND scheme has some nice theoretical results [30], it has not been widely used until the development of multilevel graph partitioning tools. State-of-the-art ordering tools [18, 36, 40, 44] are mostly a hybrid of top-down and bottom-up approaches and built using an incomplete ND approach that utilizes a multilevel graph partitioning framework [10, 35, 39, 43] for recursively identifying separators until a part becomes sufficiently small. After this point, a variant of MD, like *constraint minimum degree* (CMD) [49] is used for the ordering of the parts.

Some of these tools utilize multilevel graph partitioning by edge separator (GPES) [10, 43], whereas the others directly employ multilevel graph partitioning by vertex separator (GPVS) [40, 43]. Any edge separator found by a GPES tool can be transformed into a *wide* vertex separator by including all the vertices incident to separator edges into the vertex separator. Here, a separator is said to be *wide* if a strict subset of it forms a separator and *narrow* otherwise. The GPES-based tools utilize algorithms like vertex cover to obtain a narrow separator from this initial wide separator. It has been shown that the GPVS-based tools outperform the GPES-based tools [40], since the GPES-based tools do not directly aim to minimize vertex separator size. However, as we will demonstrate in section 2.5, GPVS-based approaches have a deficiency in the multilevel frameworks.

In this work, we propose a new incomplete ND-based fill-reducing ordering. Our approach is based on a novel formulation of the GPVS problem as a *hypergraph partitioning* (HP) problem that is immune to GPVS's deficiency in multilevel partitioning frameworks. Our formulation relies on finding an edge clique cover of the standard graph representation of matrix M. The edge clique cover is used to construct a hypergraph, which is referred to here as the clique-node hypergraph. In this hypergraph, the nodes correspond to the cliques of the edge clique cover, and the hyperedges correspond to the vertices of the standard graph representation of matrix M. We show that the partitioning of the clique-node hypergraph can be decoded as a GPVS of the standard graph representation of matrix M. In matrix terms, our formulation corresponds to finding a *structural factorization* of the matrix M in the form of

$M = AA^T$ (or $M = AD^2A^T$). Here, structural factorization refers to the fact that we are seeking a {0,1}-matrix $A = \{a_{ij}\}$, where AA^T determines the sparsity pattern of M. In applications like the solution of linear programming (LP) problems using an interior point method, such a matrix is actually given as a part of the problem. For other problems, we present efficient methods to find such a structural factorization. Furthermore, we develop matrix sparsening techniques that allow faster orderings of matrices coming from LP problems.

To the best of our knowledge, our work, including our preliminary work that had been presented in [11, 15], is the first work that utilizes hypergraph partitioning for fill-reducing ordering. This paper presents a much more detailed and formal presentation of our proposed HP-based GPVS formulation in section 3, and its application for fill-reducing ordering symmetric matrices in section 4. A recent and complementary work [34] follows a different path and tackles unsymmetric ordering by leveraging our hypergraph models for permuting matrices into singly bordered block-diagonal form [8]. The HP-based fill-reducing ordering method we introduce in section 4 is targeted for ordering symmetric matrices and uses our proposed HP-based GPVS formulation. For general symmetric matrices, the theoretical foundations of HP-based formulation of GPVS presented in this paper lead to development of two new hypergraph construction algorithms that we present in section 3.2. For matrices arising from LP problems, we present two structural factor sparsening methods in section 4.2, one of which is a new formulation of the problem as a minimum set cover problem. A detailed experimental evaluation of the proposed methods presented in section 5 shows that our method achieves better orderings in comparison to the state-of-the-art ordering tools. Finally, we conclude in section 6.

2. Preliminaries.

2.1. Graph partitioning by vertex separator. An undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is defined as a set $\mathcal{V}$ of vertices and a set $\mathcal{E}$ of edges. Every edge $e_{ij} \in \mathcal{E}$ connects a pair of distinct vertices v_i and v_j. We use the notation $Adj_{\mathcal{G}}(v_i)$ to denote the set of vertices that are adjacent to vertex v_i in graph $\mathcal{G}$. We extend this operator to include the adjacency set of a vertex subset $\mathcal{V}' \subseteq \mathcal{V}$, i.e., $Adj_{\mathcal{G}}(\mathcal{V}') = \bigcup_{v_i \in \mathcal{V}'} Adj_{\mathcal{G}}(v_i) - \mathcal{V}'$. The degree d_i of a vertex v_i is equal to the number of edges incident to v_i, i.e., $d_i = |Adj_{\mathcal{G}}(v_i)|$. A vertex subset $\mathcal{V}_S$ is a K-way *vertex separator* if the subgraph induced by the vertices in $\mathcal{V} - \mathcal{V}_S$ has at least K connected components. $\Pi_{VS} = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K; \mathcal{V}_S\}$ is a K-way vertex partition of $\mathcal{G}$ by vertex separator $\mathcal{V}_S \subseteq \mathcal{V}$ if the following conditions hold: $\mathcal{V}_k \subseteq \mathcal{V}$ and $\mathcal{V}_k \neq \emptyset$ for $1 \leq k \leq K$; $\mathcal{V}_k \cap \mathcal{V}_\ell = \emptyset$ for $1 \leq k < \ell \leq K$ and $\mathcal{V}_k \cap \mathcal{V}_S = \emptyset$ for $1 \leq k \leq K$; $\bigcup_{k=1}^{K} \mathcal{V}_k \cup \mathcal{V}_S = \mathcal{V}$; removal of $\mathcal{V}_S$ gives K disconnected parts $\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K$ (i.e., $Adj_{\mathcal{G}}(\mathcal{V}_k) \subseteq \mathcal{V}_S$ for $1 \leq k \leq K$).

In the GPVS problem, the partitioning constraint is to maintain a balance criterion on the weights of the K parts of the K-way vertex partition $\Pi_{VS} = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K; \mathcal{V}_S\}$. The weight W_k of a part $\mathcal{V}_k$ is usually defined by the number of the vertices in $\mathcal{V}_k$, i.e., $W_k = |\mathcal{V}_k|$, for $1 \leq k \leq K$. The partitioning objective is to minimize the separator size, which is usually defined as the number of vertices in the separator, i.e.,

$$Separatorsize(\Pi_{VS}) = |\mathcal{V}_S|. \tag{2.1}$$

2.2. Hypergraph partitioning. A hypergraph $\mathcal{H} = (\mathcal{U}, \mathcal{N})$ is defined as a set $\mathcal{U}$ of nodes (vertices) and a set $\mathcal{N}$ of nets (hyperedges). We refer to the vertices of $\mathcal{H}$ as nodes, to avoid the confusion between graphs and hypergraphs. Every net

$n_i \in \mathcal{N}$ connects a subset of nodes of $\mathcal{U}$, which are called the pins of n_i and are denoted as $Pins(n_i)$. The set of nets that connect node u_h is denoted as $Nets(u_h)$. Two distinct nets n_i and n_j are said to be adjacent, if they connect at least one common node. We use the notation $Adj_{\mathcal{H}}(n_i)$ to denote the set of nets that are adjacent to n_i in $\mathcal{H}$, i.e., $Adj_{\mathcal{H}}(n_i) = \{n_j \in \mathcal{N} - \{n_i\} : Pins(n_i) \cap Pins(n_j) \neq \emptyset\}$. We extend this operator to include the adjacency set of a net subset $\mathcal{N}' \subseteq \mathcal{N}$, i.e., $Adj_{\mathcal{H}}(\mathcal{N}') = \bigcup_{n_i \in \mathcal{N}'} Adj_{\mathcal{H}}(n_i) - \mathcal{N}'$. The degree d_h of a node u_h is equal to the number of nets that connect u_h, i.e., $d_h = |Nets(u_h)|$. The size s_i of a net n_i is equal to the number of its pins, i.e., $s_i = |Pins(n_i)|$.

$\Pi_{HP} = \{\mathcal{U}_1, \mathcal{U}_2, \ldots, \mathcal{U}_K\}$ is a K-way node partition of $\mathcal{H}$ if the following conditions hold: $\mathcal{U}_k \subseteq \mathcal{U}$ and $\mathcal{U}_k \neq \emptyset$ for $1 \leq k \leq K$; $\mathcal{U}_k \cap \mathcal{U}_\ell = \emptyset$ for $1 \leq k < \ell \leq K$; $\bigcup_{k=1}^{K} \mathcal{U}_k = \mathcal{U}$. In a partition Π_{HP} of $\mathcal{H}$, a net that connects at least one node in a part is said to *connect* that part. A net n_i is said to be an internal net of a node-part $\mathcal{U}_k$, if it connects only part $\mathcal{U}_k$, i.e., $Pins(n_i) \subseteq \mathcal{U}_k$. We use $\mathcal{N}_k$ to denote the set of internal nets of node-part $\mathcal{U}_k$, for $1 \leq k \leq K$. A net n_i is said to be cut (external), if it connects more than one node part. We use $\mathcal{N}_S$ to denote the set of external nets, to show that it actually forms a net separator; that is, removal of $\mathcal{N}_S$ gives at least K disconnected parts.

In the HP problem, the partitioning constraint is to maintain a balance criterion on the weights of the parts of the K-way partition $\Pi_{HP} = \{\mathcal{U}_1, \mathcal{U}_2, \ldots, \mathcal{U}_K\}$. The weight W_k of a node-part $\mathcal{U}_k$ is usually defined by the cumulative effect of the nodes in $\mathcal{U}_k$, for $1 \leq k \leq K$. However, in this work, we define W_k as the number of internal nets of node-part $\mathcal{U}_k$, i.e., $W_k = |\mathcal{N}_k|$. The partitioning objective is to minimize the cut size defined over the external nets. There are various cut-size definitions. The relevant one used in this work is the cut-net metric, where cut size is equal to the number of external nets, i.e.,

$$Cutsize(\Pi_{HP}) = |\mathcal{N}_S|. \tag{2.2}$$

2.3. Net-intersection graph representation of a hypergraph. The net-intersection graph (NIG) representation [19], also known as *intersection graph* [1, 9], was proposed and used in the literature as a fast approximation approach for solving the HP problem [41]. In the NIG representation $\text{NIG}(\mathcal{H}) = (\mathcal{V}, \mathcal{E})$ of a given hypergraph $\mathcal{H} = (\mathcal{U}, \mathcal{N})$, each vertex v_i of $\text{NIG}(\mathcal{H})$ corresponds to net n_i of $\mathcal{H}$. There exists an edge between vertices v_i and v_j of $\text{NIG}(\mathcal{H})$ if and only if the respective nets n_i and n_j are adjacent in $\mathcal{H}$, i.e., $e_{i,j} \in \mathcal{E}$ if and only if $n_j \in Adj_{\mathcal{H}}(n_i)$, which also implies that $n_i \in Adj_{\mathcal{H}}(n_j)$. This NIG definition implies that every node u_h of $\mathcal{H}$ induces a clique $\mathcal{C}_h$ in $\text{NIG}(\mathcal{H})$ where $\mathcal{C}_h = Nets(u_h)$.

2.4. Graph and hypergraph models for representing sparse matrices. Several graph and hypergraph models are proposed and used in the literature, for representing sparse matrices for a variety of applications in parallel and scientific computing [37].

In the standard graph model, a square and symmetric matrix $M = \{m_{ij}\}$ is represented as an undirected graph $\text{G}(M) = (\mathcal{V}, \mathcal{E})$. Vertex set $\mathcal{V}$ and edge set $\mathcal{E}$, respectively, correspond to the rows/columns and off-diagonal nonzeros of matrix M. There exists one vertex v_i for each row/column r_i / c_i. There exists an edge e_{ij} for each symmetric nonzero pair m_{ij} and m_{ji}; i.e., $e_{ij} \in \mathcal{E}$ if $m_{ij} \neq 0$ and $i < j$.

Three hypergraph models are proposed and used in the literature; namely, row-net, column-net, and row-column-net (a.k.a. fine-grain) hypergraph models [12, 14, 17, 53]. We will discuss only the row-net hypergraph model that is relevant to our

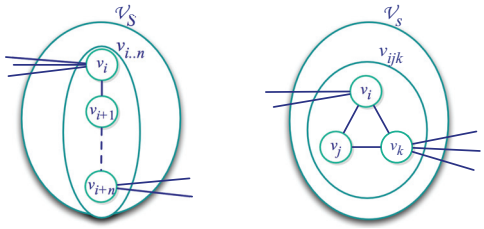


FIG. 2.1. *Partial illustration of two sample GPVS results to demonstrate the deficiency of the graph model in the multilevel framework.*

work. In the row-net hypergraph model, a rectangular matrix $A = \{a_{ij}\}$ is represented as a hypergraph $\mathrm{H_{RN}}(A) = (\mathcal{U}, \mathcal{N})$. Node set $\mathcal{U}$ and net set $\mathcal{N}$, respectively, correspond to the columns and rows of matrix A. There exist one node u_h for each column c_h and one net n_i for each row r_i. Net n_i connects the nodes corresponding to the columns that have a nonzero entry in row i; i.e., $u_h \in Pins(n_i)$ if $a_{ih} \neq 0$.

We should note that although row-net and column-net hypergraph models resemble the *bipartite graph model* [38] in structure, hypergraph models are the ones that encapsulate both the partitioning objective and the multi-interaction among vertices [37].

2.5. Deficiency of GPVS in the multilevel framework. The multilevel graph/hypergraph partitioning framework basically contains three phases: coarsening, initial partitioning, and uncoarsening. During the coarsening phase, vertices/nodes are visited in some (possibly random) order and usually two (or more) of them are coalesced to construct the vertices/nodes of the next-level coarsened graph/hypergraph. After multiple coarsening levels, an initial partition is found on the coarsest graph/hypergraph, and this partition is projected back to a partition of the original graph/hypergraph in the uncoarsening phase with further refinements at each level of uncoarsening. Both GPES and HP problems are well suited for the multilevel framework, because the following nice property holds for the edge and net separators in multilevel GPES and HP: Any edge/net separator at a given level of uncoarsening forms a valid *narrow* edge/net separator of all the finer graphs/hypergraphs, including the original graph/hypergraph. Here, an edge/net separator is said to be *narrow*, if no subset of edges/nets of the separator forms a separator.

However, this property does not hold for the GPVS problem. Consider the two examples displayed in Figure 2.1 as partial illustration of two different GPVS partitioning results at some level m of a multilevel GPVS tool. In the first example, $n+1$ vertices $\{v_i, v_{i+1}, \ldots, v_{i+n}\}$ are coalesced to construct vertex $v_{i..n}$ as a result of one or more levels of coarsening. Thus, $\mathcal{V}_S = \{v_{i..n}\}$ is a valid and narrow vertex separator for level m. The GPVS tool computes the cost of this separator as $n+1$ at this level. However, obviously this separator is a wide separator of the original graph. In other words, there is a subset of those vertices that is a valid narrow separator of the original graph. In fact, any single vertex in $\{v_i, v_{i+1}, \ldots, v_{i+n}\}$ is a valid separator of size 1 of the original graph. Similarly, for the second example, the GPVS tool computes the size of the separator as 3; however, there is a subset of constituent vertices of $\mathcal{V}_S = \{v_{ijk}\} = \{v_i, v_j, v_k\}$ that is a valid narrow separator of size 1 in the original graph. That is, either $\mathcal{V}_S = \{v_i\}$ or $\mathcal{V}_S = \{v_k\}$ is a valid narrow separator. Note that this deficiency is not because of a specific algorithm, but it is an inherent feature of the multilevel paradigm on GPVS. We refer the reader to a recent work [45]

for a more thorough comparison of GPVS and HP tools. In particular, K-way partitioning results for net balancing presented in that work experimentally confirm that a multilevel HP tool achieves smaller separator sizes than a graph-based tool.

3. HP-based GPVS formulation. We are considering a method to solve the GPVS problem for a given undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$.

3.1. Theoretical foundations. The following theorem lays down the basis for our HP-based GPVS formulation.

THEOREM 1. *Consider a hypergraph $\mathcal{H} = (\mathcal{U}, \mathcal{N})$ and its NIG representation* $\text{NIG}(\mathcal{H}) = (\mathcal{V}, \mathcal{E})$. *A K-way node-partition $\Pi_{HP} = \{\mathcal{U}_1, \mathcal{U}_2, \ldots, \mathcal{U}_K\}$ of $\mathcal{H}$ induces a K-way vertex separator $\Pi_{VS} = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K; \mathcal{V}_S\}$ of* $\text{NIG}(\mathcal{H})$, *where*

(a) *the partitioning objective of minimizing the cut size of Π_{HP} according to* (2.2) *corresponds to minimizing the separator size of Π_{VS} according to* (2.1).

(b) *the partitioning constraint of balancing on the internal net counts of node parts of Π_{HP} infers balance among the vertex counts of parts of Π_{VS}.*

Proof. As described in [8], the K-way node-partition $\Pi_{HP} = \{\mathcal{U}_1, \mathcal{U}_2, \ldots, \mathcal{U}_K\}$ of $\mathcal{H}$ induces a $(K+1)$-way net-partition $\{\mathcal{N}_1, \mathcal{N}_2, \ldots, \mathcal{N}_K; \mathcal{N}_S\}$. We consider this $(K+1)$-way net-partition $\Pi_{HP} = \{\mathcal{N}_1, \mathcal{N}_2, \ldots, \mathcal{N}_K; \mathcal{N}_S\}$ of $\mathcal{H}$ as inducing a K-way GPVS $\Pi_{VS} = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K; \mathcal{V}_S\}$ on $\text{NIG}(\mathcal{H})$, where $\mathcal{V}_k \equiv \mathcal{N}_k$, for $1 \leq k \leq K$, and $\mathcal{V}_S \equiv \mathcal{N}_S$. Consider an internal net n_j of node-part $\mathcal{U}_k$ in Π_{HP}, i.e., $n_j \in \mathcal{N}_k$. It is clear that $Adj_{\mathcal{H}}(n_j) \subseteq \mathcal{N}_k \cup \mathcal{N}_S$, which implies $Adj_{\mathcal{H}}(\mathcal{N}_k) \subseteq \mathcal{N}_S$. Since $\mathcal{V}_k \equiv \mathcal{N}_k$ and $\mathcal{V}_S \equiv \mathcal{N}_S$, $Adj_{\mathcal{H}}(\mathcal{N}_k) \subseteq \mathcal{N}_S$ in Π_{HP} implies $Adj_{\mathcal{G}}(\mathcal{V}_k) \subseteq \mathcal{V}_S$ in Π_{VS}. In other words, $Adj_{\mathcal{G}}(\mathcal{V}_k) \cap \mathcal{V}_\ell = \emptyset$, for $1 \leq \ell \leq K$ and $\ell \neq k$. Thus, $\mathcal{V}_S$ of Π_{VS} constitutes a valid separator of size $|\mathcal{V}_S| = |\mathcal{N}_S|$. So, minimizing the cut size of Π_{HP} corresponds to minimizing the separator size of Π_{VS}. Since $|\mathcal{V}_k| = |\mathcal{N}_k|$, for $1 \leq k \leq K$, balancing on the internal net counts of node parts of Π_{HP} corresponds to balancing the vertex counts of parts of Π_{VS}. ∎

COROLLARY 1. *Consider an undirected graph $\mathcal{G}$. A K-way partition Π_{HP} of any hypergraph $\mathcal{H}$ for which* $\text{NIG}(\mathcal{H}) \equiv \mathcal{G}$ *induces a K-way vertex separator Π_{VS} of $\mathcal{G}$.*

Although $\text{NIG}(\mathcal{H})$ is well defined for a given hypergraph $\mathcal{H}$, there is no unique reverse construction. We introduce the following definitions and theorems, which show our approach for reverse construction.

DEFINITION 3.1 (edge clique cover (ECC) [47]). *Given a set $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots\}$ of cliques in $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, $\mathcal{C}$ is an ECC of $\mathcal{G}$ if for each edge $e_{ij} \in \mathcal{E}$ there exists a clique $\mathcal{C}_h \in \mathcal{C}$ that contains both v_i and v_j.*

DEFINITION 3.2 (clique-node hypergraph). *Given a set $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots\}$ of cliques in graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, the clique-node hypergraph* $\text{CNH}(\mathcal{G}, \mathcal{C}) = \mathcal{H} = (\mathcal{U}, \mathcal{N})$ *of $\mathcal{G}$ for $\mathcal{C}$ is defined as a hypergraph with $|\mathcal{C}|$ nodes and $|\mathcal{V}|$ nets, where $\mathcal{H}$ contains one node u_h for each clique $\mathcal{C}_h$ of $\mathcal{C}$ and one net n_i for each vertex v_i of $\mathcal{V}$, i.e., $\mathcal{U} \equiv \mathcal{C}$ and $\mathcal{N} \equiv \mathcal{V}$. In $\mathcal{H}$, the set of nets that connect node u_h corresponds to the set $\mathcal{C}_h$ of vertices; i.e., $Nets(u_h) \equiv \mathcal{C}_h$ for $1 \leq h \leq |\mathcal{C}|$. In other words, the net n_i connects the nodes corresponding to the cliques that contain vertex v_i of $\mathcal{G}$.*

Figure 3.1(a) displays a sample graph $\mathcal{G}$ with 11 vertices and 18 edges. Figure 3.1(b) shows the clique-node hypergraph $\mathcal{H}$ of $\mathcal{G}$ for a sample ECC $\mathcal{C}$ that contains 12 cliques. Note that $\mathcal{H}$ contains 12 nodes and 11 nets. As seen in Figure 3.1(b), the 4-clique $\mathcal{C}_5 = \{v_4, v_5, v_{10}, v_{11}\}$ in $\mathcal{C}$ induces node u_5 with $Nets(u_5) = \{n_4, n_5, n_{10}, n_{11}\}$ in $\mathcal{H}$. Figure 3.2(a) shows a 3-way partition Π_{HP} of $\mathcal{H}$, where each node part contains 3 internal nets and the cut contains 2 external nets. Figure 3.2(b) shows the 3-way GPVS Π_{VS} induced by Π_{HP}. In Π_{VS}, each part contains 3 vertices and the separator contains 2 vertices. In particular, the cut with 2 external nets n_{10} and n_{11}

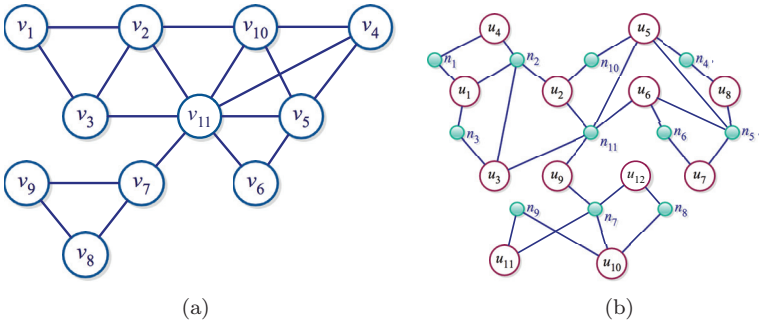


FIG. 3.1. (a) *A sample graph* $\mathcal{G}$; (b) *the clique-node hypergraph* $\mathcal{H}$ *of* $\mathcal{G}$ *for ECC* $\mathcal{C} = \{\mathcal{C}_1 = \{v_1, v_2, v_3\}, \mathcal{C}_2 = \{v_2, v_{10}, v_{11}\}, \mathcal{C}_3 = \{v_2, v_3, v_{11}\}, \mathcal{C}_4 = \{v_1, v_2\}, \mathcal{C}_5 = \{v_4, v_5, v_{10}, v_{11}\}, \mathcal{C}_6 = \{v_5, v_6, v_{11}\}, \mathcal{C}_7 = \{v_5, v_6\}, \mathcal{C}_8 = \{v_4, v_5\}, \mathcal{C}_9 = \{v_7, v_{11}\}, \mathcal{C}_{10} = \{v_7, v_8, v_9\}, \mathcal{C}_{11} = \{v_7, v_9\}, \mathcal{C}_{12} = \{v_7, v_8\}\}$.

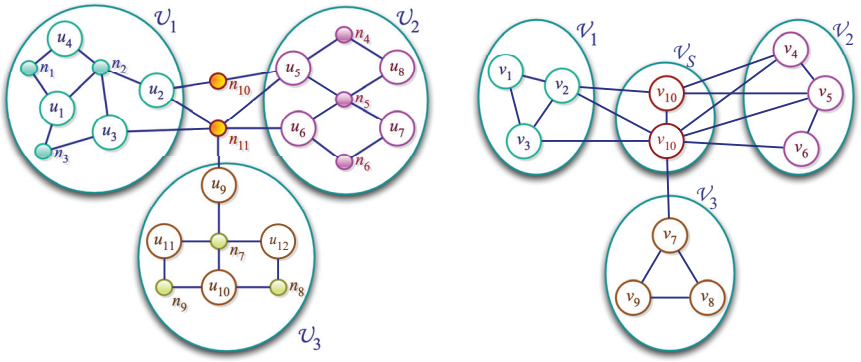


FIG. 3.2. (a) *A* 3*-way partition* Π_{HP} *of the clique-node hypergraph* $\mathcal{H}$ *given in Figure* 3.1(b); (b) *the* 3*-way GPVS* Π_{VS} *of* G *(given in Figure* 3.1(a)*) induced by* Π_{HP}.

induces a separator with 2 vertices v_{10} and v_{11}. The node-part $\mathcal{U}_1$ with 3 internal nets n_1, n_2, and n_3 induces a vertex-part $\mathcal{V}_1$ with 3 vertices v_1, v_2, and v_3.

The following two theorems state that, for a given graph $\mathcal{G}$, the problem of constructing a hypergraph whose NIG representation is the same as $\mathcal{G}$ is equivalent to the problem of finding an ECC of $\mathcal{G}$.

THEOREM 2. *Given a graph* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ *and a hypergraph* $\mathcal{H} = (\mathcal{U}, \mathcal{N})$, *if* $\text{NIG}(\mathcal{H}) \equiv \mathcal{G}$, *then* $\mathcal{H} \equiv \text{CNH}(\mathcal{G}, \mathcal{C})$ *with* $\mathcal{C} = \{\mathcal{C}_h \equiv Nets(u_h) : 1 \leq h \leq |\mathcal{U}|\}$ *is an ECC of* $\mathcal{G}$.

Proof. Since $\text{NIG}(\mathcal{H}) \equiv \mathcal{G}$, there is an edge $e_{ij} = \{v_i, v_j\}$ in $\mathcal{G}$ if and only if nets n_i and n_j are adjacent in $\mathcal{H}$, which means there exists a node u_h in $\mathcal{H}$ such that both $n_i \in Nets(u_h)$ and $n_j \in Nets(u_h)$. Since u_h induces the clique $\mathcal{C}_h \in \mathcal{C}$, $\mathcal{C}_h$ contains both vertices v_i and v_j. ∎

Note that $\mathcal{C} = \{\mathcal{C}_h \equiv Nets(u_h) : 1 \leq h \leq |\mathcal{U}|\}$ is the unique ECC of $\mathcal{G}$ satisfying $\mathcal{H} \equiv \text{CNH}(\mathcal{G}, \mathcal{C})$.

THEOREM 3. *Given a graph* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, *for any ECC* $\mathcal{C}$ *of* $\mathcal{G}$, *the NIG representation of the clique-node hypergraph of* $\mathcal{C}$ *is equivalent to* $\mathcal{G}$, *i.e.,* $\text{NIG}(\text{CNH}(\mathcal{G}, \mathcal{C})) \equiv \mathcal{G}$.

Proof. By construction, two nets n_i and n_j are adjacent in $\mathrm{CNH}(\mathcal{G},\mathcal{C})$ if and only if there exists a clique $\mathcal{C}_h \in \mathcal{C}$ such that $\mathcal{C}_h$ contains both vertices v_i and v_j in $\mathcal{G}$. Since $\mathcal{C}$ is an ECC of $\mathcal{G}$, there is such a clique $\mathcal{C}_h \in \mathcal{C}$ if and only if there is an edge e_{ij} in $\mathcal{G}$. ∎

3.2. Hypergraph construction based on edge clique cover. According to the theoretical findings given in section 3.1, our HP-based GPVS approach is based on finding an ECC of the given graph and then partitioning the respective clique-node hypergraph. Here, we will briefly discuss the effects of different ECCs on the solution quality and the run-time performance of our approach.

In terms of solution quality of hypergraph partitioning, it is not easy to quantify the metrics for a "good" ECC. In a multilevel HP tool that balances internal net weights, the choice of an ECC should not affect the quality performance of the FM-like [27] refinement heuristics commonly used in the uncoarsening phase. However, the choice of an ECC may considerably affect the quality performance of the node matchings performed in the coarsening phase. For example, large cliques in the ECC may lead to better quality node matchings even in the initial coarsening levels. On the other side, large amounts of edge overlaps among the cliques of a given ECC may adversely affect the quality of the node matchings. Therefore, having large but nonoverlapping cliques might be desirable for solution quality.

The choice of the ECC may affect the run-time performance of the HP tool depending on the size of the clique-node hypergraph. Since the number of nets in the clique-node hypergraph is fixed, the number of cliques and the sum of the clique sizes, which, respectively, correspond to the number of nodes and pins, determine the size of the hypergraph. Hence, an ECC with a small number of large cliques is likely to induce a clique-node hypergraph of small size.

Although not a perfect match, the ECC problem [47], which is stated as finding an ECC with minimum number of cliques, can be considered to be relevant to our problem of finding a "good" ECC. Unfortunately, the ECC problem is also known to be NP-hard [47]. The literature contains a number of heuristics [33, 46, 47] for solving the ECC problem. However, even the fastest heuristic's [33] running time complexity is $O(|\mathcal{V}||\mathcal{E}|)$, which makes it impractical in our approach.

In this work, we investigate three different types of ECCs, namely, $\mathcal{C}^2$, $\mathcal{C}^3$, and $\mathcal{C}^4$, to observe the effects of increasing clique size in the solution quality and run-time performance of the proposed approach. Here, $\mathcal{C}^2$ denotes the ECC of all 2-cliques (edges), i.e., $\mathcal{C}^2 = \mathcal{E}$; $\mathcal{C}^3$ denotes an ECC of 2- and 3-cliques; $\mathcal{C}^4$ denotes an ECC of 2-, 3-, and 4-cliques. In general, $\mathcal{C}^k$ denotes an ECC of cliques in which maximum clique size is bounded above by k. Note that $\mathcal{C}^2$ is unique, whereas $\mathcal{C}^3$ and $\mathcal{C}^4$ are not necessarily unique. We will refer to the clique-node hypergraph induced by $\mathcal{C}^k$ as $\mathcal{H}^k = \mathrm{CNH}(\mathcal{G},\mathcal{C}^k)$.

The clique-node hypergraph $\mathcal{H}^2$ deserves special attention, since it is uniquely defined for a given graph $\mathcal{G}$. In $\mathcal{H}^2$, there exists one node of degree 2 for each edge e_{ij} of $\mathcal{G}$. The net n_i corresponding to vertex v_i of $\mathcal{G}$ connects all nodes corresponding to the edges that are incident to vertex v_i, for $1 \le i \le |\mathcal{V}|$. So, $\mathcal{H}^2$ contains $|\mathcal{E}|$ nodes, $|\mathcal{V}|$ nets, and $2|\mathcal{E}|$ pins. The running time of HP-based GPVS using $\mathcal{H}^2$ is expected to be quite high because of the large number of nodes and pins. Figure 3.3 displays the 2-clique-node hypergraph $\mathcal{H}^2$ of the sample graph $\mathcal{G}$ given in Figure 3.1(a). As seen in the figure, each node of $\mathcal{H}^2$ is labeled as u_{ij} to show the one-to-one correspondence between nodes of $\mathcal{H}^2$ and edges of $\mathcal{G}$. That is, node u_{ij} of $\mathcal{H}^2$ corresponds to edge e_{ij} of $\mathcal{G}$, where $Nets(u_{ij}) = \{n_i, n_j\}$.

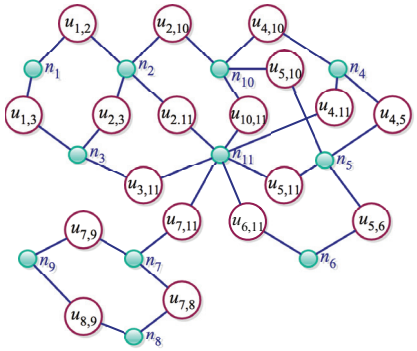


FIG. 3.3. *The 2-clique-node hypergraph $\mathcal{H}^2$ of graph $\mathcal{G}$ given in Figure* 3.1(a).

Algorithm 1. $\mathcal{C}^3$ Construction Algorithm

```
Data: G = (V, E)
for each vertex v ∈ V do
    π1[v] ← NIL
for each edge e_ij ∈ E do
    cover[e_ij] ← 0
C^3 ← ∅
for each vertex v_i ∈ V do
    for each vertex v_j ∈ Adj_G(v_i) with j > i do
        π1[v_j] ← v_i
    for each vertex v_j ∈ Adj_G(v_i) with j > i do
        for each vertex v_k ∈ Adj_G(v_j) with k > j do
            if π1[v_k] = v_i then
                if Σ_{e ∈ C(2, {v_i, v_j, v_k})} cover[e] < 2 then
                    C^3 ← C^3 ∪ {{v_i, v_j, v_k}}        ▷ Add the 3-clique to C^3
                    for each edge e ∈ C(2, {v_i, v_j, v_k}) do
                        cover[e] ← 1
        if cover[e_ij] = 0 then
            C^3 ← C^3 ∪ {{v_i, v_j}}                    ▷ Add the 2-clique to C^3
            cover[e_ij] ← 1
```

Algorithm 1 displays the algorithm developed for constructing a $\mathcal{C}^3$, whereas the algorithm developed for constructing a $\mathcal{C}^4$ is given in our technical report [16]. The goal of both algorithms is to minimize the number of pins in the clique-node hypergraphs as much as possible. Both algorithms visit the vertices in random order in order to introduce randomization to the ECC construction process. In both algorithms, each edge is processed along only one direction (i.e., from low to high numbered vertex) to avoid identifying the same clique more than once.

In Algorithm 1, for each visited vertex v_i, 3-cliques that contain v_i are searched for by trying to locate 2-cliques between the vertices in $Adj_{\mathcal{G}}(v_i)$. This search is performed by scanning the adjacency list of each vertex v_j in $Adj_{\mathcal{G}}(v_i)$. For each vertex, a parent field π_1 is maintained for efficient identification of 3-cliques during

this search. An identified 3-clique C_h is selected for inclusion in $\mathcal{C}^3$ if the number of already covered edges of C_h is at most 1. The rationale behind this selection criterion is as follows: Recall that a 3-clique in $\mathcal{C}^3$ adds 3 pins to $\mathcal{H}^3$, since it incurs a node of degree 3 in $\mathcal{H}^3$. If only one edge of C_h is already covered by an other 3-clique in $\mathcal{C}^3$, it is still beneficial to cover the remaining two edges of C_h by selecting C_h instead of selecting the two 2-cliques covering those uncovered edges, because the former selection incurs 3 pins, whereas the latter incurs 4 pins. If, however, any two edges of C_h are already covered by another 3-clique in $\mathcal{C}^3$, it is clear that the remaining uncovered edge is better to be covered by a 2-clique. After scanning the adjacency list of v_j in $Adj_{\mathcal{G}}(v_i)$, if edge $\{v_i, v_j\}$ is not covered by any 3-clique, which is detected by holding a cover field for each edge where cover$[e]$ is a boolean that registers whether or not the edge e is covered already, then it is added to $\mathcal{C}^3$ as a 2-clique. Algorithm 1 runs in $O(|\mathcal{V}|\Delta^2)$ time where Δ denotes the maximum degree of $\mathcal{G}$.

The $\mathcal{C}^4$-construction algorithm, the details of which can be found in [16], runs in $O(|\mathcal{V}|\Delta^3)$-time. We should note here that the ideas in the $\mathcal{C}^3$- and $\mathcal{C}^4$-construction algorithms can be extended to a general approach for constructing $\mathcal{C}^k$. However, this general approach requires maintaining $k-2$ parent fields for each vertex and runs in $O(|\mathcal{V}|\Delta^{k-1})$ time.

3.3. Matrix-theoretic view of HP-based GPVS formulation. Here, we will try to reveal the association between the graph-theoretic and matrix-theoretic views of our HP-based GPVS formulation. Given a $p \times p$ symmetric and square matrix M, let $\mathrm{G}(M) = (\mathcal{V}, \mathcal{E})$ denote the standard graph representation of matrix M.

A K-way GPVS $\Pi_{VS} = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K; \mathcal{V}_S\}$ of $\mathrm{G}(M)$ can be decoded as permuting matrix M into a doubly bordered block diagonal (DB) form $M_{DB} = PAP^T$ as follows: Π_{VS} is used to define the partial row/column permutation matrix P by permuting the rows/columns corresponding to the vertices of $\mathcal{V}_k$ after those corresponding to the vertices of $\mathcal{V}_{k-1}$ for $2 \leq k \leq K$, and permuting the rows/columns corresponding to the separator vertices to the end. The partitioning objective of minimizing the separator size of Π_{VS} corresponds to minimizing the number of coupling rows/columns in M_{DB}, whereas the partitioning constraint of maintaining balance on the part weights of Π_{VS} infers balance among the row/column counts of the square diagonal submatrices in M_{DB}.

In the graph-theoretic discussion given in section 3.2, we are looking for a hypergraph $\mathcal{H}$ whose NIG representation is equivalent to $\mathrm{G}(M)$. In matrix-theoretic view, this corresponds to looking for a structural factorization $M = AA^T$ of matrix M, where A is an $p \times q$ rectangular matrix. Here, structural factorization refers to the fact that $A = \{a_{ij}\}$ is a {0,1}-matrix, where AA^T determines the sparsity patterns of M. In this factorization, the rows of matrix A correspond to the vertices of $\mathrm{G}(M)$ and the set of columns of matrix A determines an ECC $\mathcal{C}$ of $\mathrm{G}(M)$. So, matrix A can be considered as a clique incidence matrix of $\mathrm{G}(M)$. That is, column c_h of matrix A corresponds to a clique $\mathcal{C}_h$ of $\mathcal{C}$, where $a_{ih} \neq 0$ implies that vertex $v_i \in \mathcal{C}_h$. The row-net hypergraph model $\mathrm{H}_{\mathrm{RN}}(A)$ of matrix A is equivalent to the clique-node hypergraph of graph $\mathrm{G}(M)$ for the ECC $\mathcal{C}$ determined by the columns of A, i.e., $\mathrm{H}_{\mathrm{RN}}(A) \equiv \mathrm{CNH}(\mathrm{G}(M), \mathcal{C})$. In other words, the NIG representation of row-net hypergraph model $\mathrm{H}_{\mathrm{RN}}(A)$ of matrix A is equivalent to $\mathrm{G}(M)$, i.e., $\mathrm{NIG}(\mathrm{H}_{\mathrm{RN}}(A)) \equiv \mathrm{G}(M)$.[1]

[1] We would like to note the relation of net intersection graph with *column intersection graph* [31]. The column intersection graph of a given matrix A is equal to the net intersection graph of the column-net hypergraph representation of A.

As shown in [8], a K-way node-partition $\Pi_{HP} = \{\mathcal{U}_1, \mathcal{U}_2, \ldots, \mathcal{U}_K\}$, which induces a $(K+1)$-way net partition $\{\mathcal{N}_1, \mathcal{N}_2, \ldots, \mathcal{N}_K; \mathcal{N}_S\}$, of $\mathrm{H}_{\mathrm{RN}}(A)$ can be decoded as permuting matrix A into a K-way rowwise singly bordered block diagonal (SB) form

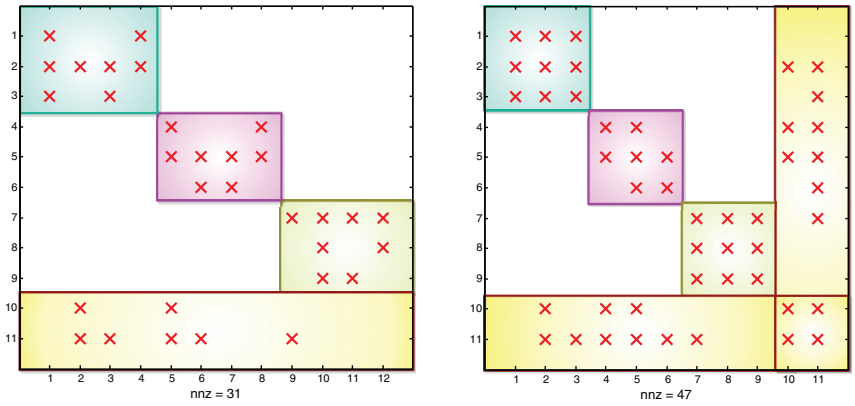
$$A_{SB} = PAQ = \begin{bmatrix} A_1 & & \\ & \ddots & \\ & & A_K \\ A_{B_1} & \ldots & A_{B_K} \end{bmatrix}. \tag{3.1}$$

Here, the K-way node partition is used to define the partial column permutation matrix Q by permuting the columns corresponding to the nodes of part $\mathcal{U}_k$ after those corresponding to the nodes of part $\mathcal{U}_{k-1}$ for $2 \leq k \leq K$. The $(K+1)$-way partition on the nets of $\mathrm{H}_{\mathrm{RN}}(A)$ is used to define the partial row permutation matrix P by permuting the rows corresponding to the nets of $\mathcal{N}_k$ after those corresponding to the nets of $\mathcal{N}_{k-1}$ for $2 \leq k \leq K$, and permuting the rows corresponding to the external nets to the end. Here, the partitioning objective of minimizing the cut size of Π_{HP} corresponds to minimizing the number of coupling rows in A_{SB}. The partitioning constraint of balancing on the internal net counts of node parts of Π_{HP} infers balance among the row counts of the rectangular diagonal submatrices in A_{SB}. It is clear that the transpose of A_{SB} will be in a columnwise SB form.

An SB form A_{SB} of A induces a DB form M_{DB} of M, since multiplying A_{SB} with its transpose produces a DB form of M [28]. That is,

$$\begin{aligned} A_{SB}A_{SB}^T &= \begin{bmatrix} A_1 & & \\ & \ddots & \\ & & A_K \\ A_{B_1} & \ldots & A_{B_K} \end{bmatrix} \begin{bmatrix} A_1^T & & & A_{B_1}^T \\ & \ddots & & \vdots \\ & & A_K^T & A_{B_K}^T \end{bmatrix} \\ &= \begin{bmatrix} A_1A_1^T & & & A_1A_{B_1}^T \\ & \ddots & & \vdots \\ & & A_KA_K^T & A_KA_{B_K}^T \\ A_{B_1}A_1^T & \ldots & A_{B_K}A_K^T & \sum_k A_{B_k}A_{B_k}^T \end{bmatrix} = M_{DB}. \end{aligned} \tag{3.2}$$

As seen in (3.2), the number of rows/columns in the square diagonal block $A_kA_k^T$ of M_{DB} is equal to the number of rows of the rectangular diagonal block A_k of A_{SB}. Furthermore, the number of coupling rows/columns in M_{DB} is equal to the number of coupling rows in A_{SB}. So, minimizing the number of coupling rows in A_{SB} corresponds to minimizing the number of coupling rows/columns in M_{DB}, whereas balancing on row counts of the rectangular diagonal submatrices in A_{SB} infers balance among the row/column counts of the square diagonal submatrices in M_{DB}. Thus, given a structural factorization $M = AA^T$ of matrix M, the proposed HP-based GPVS formulation corresponds to formulating the problem of permuting M into a DB block diagonal form as an instance of the problem of permuting A into an SB block diagonal form. Figure 3.4 shows the matrix theoretical view of our HP-based GPVS formulation on the sample graph, hypergraph, and their partitions given in Figures 3.1 and 3.2.



(a) (b)

FIG. 3.4. (a) *Matrix A whose row-net hypergraph representation is given in Figure* 3.1(b) *and its* 3*-way SB form A_{SB} induced by the* 3*-way partition Π_{HP} given in Figure* 3.2(a); (b) *matrix M whose standard graph representation is given in Figure* 3.1(a) *and its* 3*-way DB form M_{DB} induced by A_{SB}.*

4. HP-based fill-reducing ordering. Given a $p \times p$ symmetric and square matrix $M = \{m_{ij}\}$ for fill-reducing ordering, let $\mathrm{G}(M) = (\mathcal{V}, \mathcal{E})$ denote the standard graph representation of matrix M.

4.1. Incomplete-nested-dissection-based orderings via recursive hypergraph bipartitioning. As described in [7], the fill-reducing matrix reordering schemes based on incomplete nested dissection can be classified as ND and *multisection* (MS). Both schemes apply 2-way GPVS (bisection) recursively on $\mathrm{G}(M)$ until the parts (domains) become fairly small. After each bisection step, the vertices in the 2-way separator (bisector) are removed and the further bisection operations are recursively performed on the subgraphs induced by the parts of bisection. In the proposed recursive-HP-based ordering approach, the constructed hypergraph $\mathcal{H}$ (where $\mathrm{NIG}(\mathcal{H}) \equiv \mathrm{G}(M)$) is bipartitioned recursively until the number of internal nets of the parts become fairly small. After each bipartitioning step, the cut nets are removed and the further bipartitioning operations are recursively performed on the subhypergraphs induced by the node parts of the bipartition. Note that this cut-net removal scheme in recursive 2-way HP corresponds to the above-mentioned separator-vertex removal scheme in recursive 2-way GPVS.

As mentioned above, both ND and MS schemes effectively obtain a multiway separator (multisector) at the end of the recursive 2-way GPVS operations. In both schemes, the parts of the multiway separator are ordered using an MD-based algorithm before the separator. It is clear that the parts can be ordered independently. These two schemes differ in the order that they number the vertices of the multiway separator. In the ND scheme, the 2-way separators constituting the multiway separator are numbered using an MD-based algorithm in depth-first order of the recursive bisection process. Note that the 2-way separators at the same level of the recursive bisection tree can be ordered independently. In the MS scheme, the multiway separator is ordered using an MD-based algorithm as a whole in a single step.

Figure 4.1 displays a sample 4-way SB form of a matrix A and the corresponding 4-way DB form of the corresponding matrix M induced by a 2-level recursive bipar-

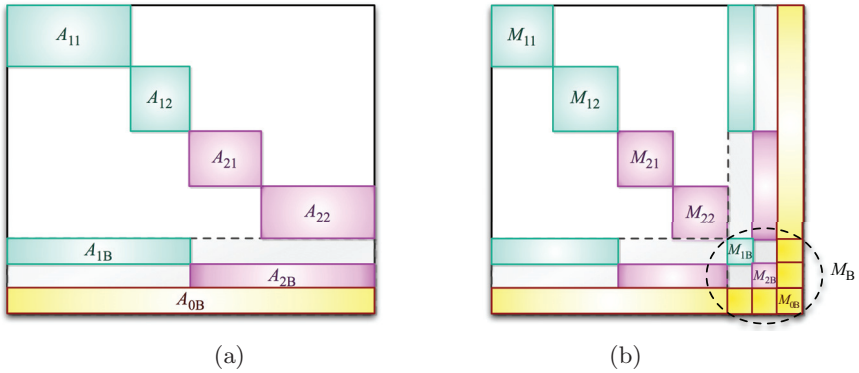


FIG. 4.1. (a) *A sample* 4*-way SB form of a matrix* A *obtained through a* 2*-level recursive hypergraph bipartitioning process;* (b) *the corresponding* 4*-way DB form of matrix* $M = AA^T$.

titioning/bisection process. Here, the bipartitioning/bisection operation at the root level is numbered as 0, whereas the bipartitioning/bisection operations at the second level are numbered as 1 and 2. The parts of a bipartition/bisection are always numbered as 1 and 2, whereas the border is numbered as B. For example, A_{11}/M_{11} and A_{12}/M_{12} denote the diagonal domain submatrices corresponding to the two parts of the bipartitioning/bisection operation 1, whereas A_{21}/M_{21} and A_{22}/M_{22} denote the diagonal domain submatrices corresponding to the two parts of the bipartitioning/bisection operation 2. As seen in the figure, $M_{0B} = A_{0B}A_{0B}^T$ denotes the diagonal border submatrix corresponding to the 2-way separator obtained at the root level, whereas $M_{1B} = A_{1B}A_{1B}^T$ and $M_{2B} = A_{2B}A_{2B}^T$ denote the diagonal border submatrices corresponding to the 2-way separators obtained at the second level. Note that M_B denotes the diagonal border submatrix corresponding to the overall 4-way separator. In both ND and MS schemes, diagonal domain submatrices are ordered before the diagonal border submatrix M_B. In the ND scheme, diagonal border submatrices are ordered in depth-first order M_{1B}, M_{2B}, and M_{0B} of the recursive bisection process. In the MS scheme, the overall diagonal border submatrix M_B is ordered as a whole.

4.2. Structural factor sparsening for ordering LP matrices. Interior point methods are widely used for solving linear programming problems [21]. These are iterative methods and usually adopt the normal equations approach [4]. The main computational cost at each iteration is the solution of a symmetric positive definite system of the form $Mx = b$, where $M = AD^2A^T$. Here, $A = \{a_{ij}\}$ is a $p \times q$ sparse rectangular constraint matrix that remains constant throughout the iterations, and D^2 is a $q \times q$ diagonal scaling matrix that changes from iteration to iteration. This linear system is typically solved by computing the Cholesky factorization ($M = LL^T$) of M, and solving the triangular system through forward and backward substitution. So, fill-reducing ordering of matrix M is crucial in the overall performance of the interior point algorithm.

Since D^2 is a diagonal matrix, AA^T determines the sparsity pattern of M. So, by neglecting numerical cancellations that may occur in matrix-matrix-transpose multiplication AA^T, we can consider $A = \{a_{ij}\}$ as a $\{0,1\}$-matrix so that $M = AA^T$ gives us a structural factorization of matrix M. Note that the matrix A may contain redundant columns and/or nonzeros in terms of determining the sparsity pattern of M. Here, we will propose and discuss two matrix sparsening algorithms that aim

at deleting as many columns and/or nonzeros of matrix A without disturbing the sparsity pattern of matrix M. The objective is to speed up the proposed HP-based GPVS method for ordering LP matrices through decreasing the size of the row-net hypergraph representation of matrix A. Both algorithms consider both column and nonzero deletions. However, the first algorithm is nonzero-deletion based, whereas the second one is column-deletion based.

For the nonzero-deletion-based sparsening algorithm, we define b_{ij} to denote the number of common columns between rows r_i and r_j of matrix A. A column c_h is said to be common between rows r_i and r_j if both rows have a nonzero in column c_h. Note that b_{ij} is equal to the integer value of nonzero m_{ij} of matrix M if $M = AA^T$ is computed using A as a {0,1}-matrix. So, the sparsity pattern of M will remain the same as long as b_{ij} values corresponding to the nonzeros of matrix M remain greater than or equal to 1 during nonzero deletions in matrix A. In particular, a nonzero a_{ih} of matrix A can be deleted if $b_{ij} > 1$ for each nonzero a_{jh} in column c_h of matrix A.

The proposed nonzero-deletion-based sparsening algorithm, spNZ, is given in Algorithm 2. Note that the quality of the sparsening depends on the processing order of nonzeros for deletion. Algorithm 2 considers the nonzeros for deletion in row major order. In the doubly nested for loop in lines 4–6, the b_{ij} values for row r_i are computed in 1D array B. Then, for each nonzero a_{ih} in row r_i, the for loop in lines 9–12 checks whether the condition $b_{ij} > 1$ holds for each nonzero a_{jh} in column c_h

Algorithm 2. spNZ: Nonzero-Deletion-Oriented Sparsening Algorithm

```
   Data: A: both in CSR and CSC formats
 1 for each row r_i ∈ A do
 2     B[i] ← 0
 3 for each row r_i ∈ A do
 4     for each nonzero a_ih ∈ r_i do
 5         for each nonzero a_jh ∈ c_h do
 6             B[j] ← B[j]+1
 7     for each nonzero a_ih ∈ r_i do
 8         flag ← TRUE
 9         for each nonzero a_jh ∈ c_h do
10             if B[j] = 1 then
11                 flag ← FALSE
12                 break
13         if flag = TRUE then
14             for each nonzero a_jh ∈ c_h do
15                 B[j] ← B[j]−1
16             delete nonzero a_ih
17     for each nonzero a_ih ∈ r_i do
18         for each nonzero a_jh ∈ c_h do
19             B[j] ← 0
20 for each column c_h ∈ A do
21     if c_h is empty then
22         delete column c_h
```

of matrix A. If it is so, the relevant b_{ij} (i.e., B_j) values are decremented and the nonzero a_{ih} is deleted in lines 13–16. At the end of the algorithm, the columns that become empty due to the nonzero deletions are detected and deleted by the for loop in lines 20–22. This algorithm runs in $O(\sum_{c_h \in A} |c_h|^2)$ time, where $|c_h|$ denotes the number of nonzeros in column c_h.

In the column-deletion-based sparsening, the objective is to maximize the number of A-matrix column deletions without disturbing the sparsity pattern of matrix M. This problem can be formulated as a minimum set cover problem as follows: The set of nonzeros of matrix M constitutes the main set of elements, whereas the set of A-matrix columns constitutes a family $\mathcal{F}$ of subsets of the main set. For each A-matrix column c_h, the subset $\mathrm{S}(c_h)$ of the main set of elements is defined as $\mathrm{S}(c_h) = \{m_{ij} \in M : a_{ih} \text{ and } a_{jh} \text{ are nonzeros}\}$. That is, each nonzero pair (a_{ih}, a_{jh}) in column c_h contributes m_{ij} to the subset $\mathrm{S}(c_h)$. The objective of the minimum set cover problem is to find a minimum number of subsets covering the main set. This objective corresponds to minimizing the number of A-matrix columns to be retained (maximizing the number of A-matrix columns to be deleted) without disturbing the sparsity pattern of matrix M.

The minimum set cover problem is known to be NP-hard [42]. However, there is a well-known $(\ln n)$-approximation algorithm [20]. A two-phase sparsening algorithm, which we will call spCol, is developed based on this minimum set cover algorithm as follows: In the first phase, the set cover algorithm is used to obtain a matrix A_c whose columns correspond to a minimum set of A-matrix columns that covers the set of all nonzeros of M. In the second phase, Algorithm 2 is run on matrix A_c for nonzero deletions.

5. Experimental results. The proposed HP-based GPVS formulation is embedded into the state-of-the-art HP tool PaToH [13], and the resulting HP-based fill-reducing ordering tool is referred to here as oPaToH. In oPaToH, the recursive hypergraph bipartitioning process is continued until the number of internal nets of a part of a bipartition drops below 200 or the number of nodes of a part of a bipartition drops below 100. oPaToH implements both MS and ND schemes; for the sake of simplicity in the presentation we will present only ND scheme results in this paper. oPaToH uses SMOOTH's [6] implementation of the CMD [49] algorithm for ordering decoupled diagonal domain submatrices and the MMD [48] algorithm for ordering diagonal border submatrices.

The performance of oPaToH is compared against the state-of-the-art ordering algorithms and tools MeTiS [44], AMD [3], COLAMD [23], and SMOOTH [6].[2] MeTiS v4.0 [44] provides two multilevel nested dissection [43] programs: oemetis and onmetis, which are GPES based and GPVS based, respectively. GPVS-based ordering in general performs better than GPES-based ordering [40], and since our earlier experiments, using the test matrices of this study, comply with this fact, we are only presenting the onmetis results here, for the sake of simplicity in the presentation. The onmetis uses MMD for ordering decoupled diagonal domain submatrices and diagonal border submatrices. We present the results for SMOOTH that utilizes the MS scheme. oPaToH uses CMD for ordering decoupled diagonal domain submatrices and MMD for ordering diagonal border submatrices. All the codes were run on a 24-core PC equipped with quad 2.1Ghz 6-core AMD Opteron processors with 6 128 KB L1

[2]The SMOOTH sparse matrix ordering package has later been included in the sparse linear system solver package called SPOOLES [5]. We will continue to use the name SMOOTH to denote that we are referring to the ordering package.

and 512 KB L2 caches, and a single 6MB L3 cache. The system is 128 GB memory and runs Debian Linux v5.0.5.

We performed experimental evaluation of the proposed HP-based fill-reducing ordering approach using 50 matrices obtained from the University of Florida sparse matrix collection [22]. The first 25 matrices are general symmetric and square matrices M arising in different application domains, mostly discretization on regular 2D/3D grids, whereas the remaining 25 M matrices are derived from LP constraint matrices using $M = AA^T$. Table 5.1 illustrates the properties of these matrices. In this table, p and $nnz(M)$ denote, respectively, the number of rows/columns and nonzeros of matrix M. For a matrix M derived from an LP problem, the number of columns q and nonzeros $nnz(A)$ are also listed for the respective A-matrix. Note that the number of rows of A is equal to the number of rows/columns of M. The general matrices are further divided into three groups (first 5, second 5, and remaining 15) according to the size of the maximum cliques that can be obtained from their graph representations. The reason for this division will become clear during the discussion of Table 5.2. The matrices in each category/group are listed in increasing order of number of nonzeros. This table also displays the performance of the onmetis ordering in terms of operation count in triangular factorization (shown as *opc*), number of nonzeros in the triangular factor (shown as $nnz(L)$), and ordering time in seconds.

The detailed performance comparison of nonzero-deletion-based (spNZ) and column-deletion-based (spCol) matrix sparsening algorithms are reported in our technical report [16]. We summarize this detailed performance comparison as follows. In terms of the ordering quality, oPaToH using spNZ and oPaToH using spCol display very close performance to that of oPaToH using the original A-matrix. Both sparsening algorithms amortize the sparsening overhead by considerably reducing the ordering time such that oPaToH using spNZ and oPaToH using spCol, respectively, run 18% and 10% faster than oPaToH using the original A matrix, on the average. Therefore, spNZ is used for sparsening in oPaToH for LP matrices.

Table 5.2 displays the properties of the hypergraphs in terms of number of nodes and pins. In the table, $\mathcal{H}^2$, $\mathcal{H}^3$, and $\mathcal{H}^4$ denote the clique-node hypergraphs induced by ECCs $\mathcal{C}^2$, $\mathcal{C}^3$, and $\mathcal{C}^4$, respectively. For LP matrices, $\mathrm{H}_{\mathrm{RN}}(\tilde{A})$ refers to the hypergraphs obtained from row-net representations of the sparsened A matrices. Note that, for ordering LP matrices, we recommend to use $\mathrm{H}_{\mathrm{RN}}(\tilde{A})$ hypergraphs. Here, we provide the results for $\mathcal{H}^2$, $\mathcal{H}^3$, and $\mathcal{H}^4$ hypergraphs. Also note that, for a given matrix M, all hypergraphs have the same number of nets, which is equal to the number of rows/columns of M. In the table, the $\mathcal{H}^2$ model is considered as the base model, so the number of nodes and pins of $\mathcal{H}^3$, $\mathcal{H}^4$, and $\mathrm{H}_{\mathrm{RN}}(\tilde{A})$ are displayed as normalized with respect to those of $\mathcal{H}^2$.

As seen in Table 5.2, the size of the clique-node hypergraph for a given matrix M decreases in terms of both number of nodes and number of pins when larger cliques of $\mathrm{G}(M)$ are considered while constructing the hypergraph. That is, $\mathcal{H}^4$ has smaller size than $\mathcal{H}^3$, which in turn has smaller size than $\mathcal{H}^2$. However, the first 5 and the first 10 out of 25 general matrices do not lead to 3-cliques and 4-cliques, respectively. So, the $\mathcal{H}^2$, $\mathcal{H}^3$, and $\mathcal{H}^4$ hypergraphs are the same for the first 5 general matrices M, whereas the $\mathcal{H}^3$ and $\mathcal{H}^4$ hypergraphs are the same for the first 10 general matrices M. As seen in Table 5.2, for LP matrices, $\mathrm{H}_{\mathrm{RN}}(\tilde{A})$ hypergraphs have drastically smaller size than even $\mathcal{H}^4$ hypergraphs in general. We should note here that the memory footprint of graph- and hypergraph-based ordering tools will be proportional to the size of the graph and hypergraph they are operating on, respectively. The memory

TABLE 5.1
Properties of test matrices and results of onmetis orderings.

Name	$p \times p$ matrix M		$p \times q$ matrix A		Onmetis		
	p	$nnz(M)$	q	$nnz(A)$	opc	$nnz(L)$	Time (s)
General M matrices							
ncvxqp9	16,554	61,540	—	—	6.33E+06	140,016	0.080
aug3dcqp	35,543	136,115	—	—	2.88E+08	1,057,586	0.280
c-53	30,235	372,213	—	—	3.68E+07	434,369	0.330
c-59	41,282	480,536	—	—	2.73E+09	3,476,329	0.520
c-67	57,975	531,935	—	—	1.27E+07	486,890	0.620
lshp3025	3,025	20,833	—	—	3.23E+06	75,083	0.010
lshp3466	3,466	23,896	—	—	3.91E+06	87,804	0.010
bodyy4	17,546	121,938	—	—	3.44E+07	519,040	0.090
rail_20209	20,209	139,233	—	—	1.41E+07	339,610	0.130
cvxbqp1	50,000	349,968	—	—	4.94E+08	2,073,553	0.340
shuttle_eddy	10,429	103,599	—	—	2.23E+07	363,205	0.060
nasa4704	4,704	104,756	—	—	3.87E+07	301,609	0.020
bcsstk24	3,562	159,910	—	—	4.20E+07	316,582	0.010
skirt	12,598	196,520	—	—	3.11E+07	483,714	0.090
bcsstk28	4,410	219,024	—	—	5.52E+07	407,462	0.010
s1rmq4m1	5,489	281,111	—	—	1.09E+08	652,367	0.010
vibrobox	12,328	342,828	—	—	1.01E+09	2,214,711	0.170
crystk01	4,875	315,891	—	—	2.76E+08	1,011,036	0.020
bcsstm36	23,052	331,486	—	—	1.17E+08	902,765	0.240
gridgena	48,962	512,084	—	—	3.61E+08	2,700,347	0.400
k1_san	67,759	580,579	—	—	4.14E+08	2,666,745	0.650
finan512	74,752	596,992	—	—	1.52E+08	1,794,080	0.650
msc23052	23,052	1,154,814	—	—	6.48E+08	2,957,144	0.050
bcsstk35	30,237	1,450,163	—	—	5.15E+08	3,116,057	0.100
oilpan	73,752	3,597,188	—	—	2.81E+09	9,211,195	0.140
Linear programming matrices $M = AA^T$							
lp_pds_02	2,953	23,281	7,716	16,571	1.92E+06	44,788	0.020
delf	3,170	33,508	6,654	15,397	1.92E+06	53,355	0.020
lp_dfl001	6,071	82,267	12,230	35,632	7.23E+08	1,254,715	0.060
model9	2,879	103,961	10,939	55,956	5.36E+06	101,358	0.040
nl	7,039	105,089	15,325	47,035	4.19E+07	302,932	0.060
ge	10,099	112,129	16,369	44,825	2.46E+07	279,501	0.080
nemsemm2	6,943	145,413	48,878	182,012	6.31E+06	149,308	0.070
lp_nug12	3,192	152,376	8,856	38,304	3.36E+09	2,566,910	0.060
lp_ken_13	28,632	161,804	42,659	97,246	1.83E+07	378,309	0.150
lpi_gosh	3,792	206,010	13,455	99,953	3.98E+07	260,364	0.050
cq9	9,278	221,590	21,534	96,653	4.28E+07	418,398	0.090
lp_osa_14	2,337	230,023	54,797	317,097	6.56E+06	118,497	0.060
co9	10,789	249,205	22,924	109,651	5.59E+07	496,545	0.090
pltexpa	26,894	269,736	70,364	143,059	1.88E+08	1,305,653	0.240
model10	4,400	293,260	16,819	150,372	5.75E+07	394,819	0.070
fome12	24,284	329,068	48,920	142,528	2.86E+09	4,999,922	0.330
lp_cre_d	8,926	372,266	73,948	246,614	2.10E+08	761,732	0.180
r05	5,190	406,158	9,690	104,145	1.22E+08	533,825	0.070
p010	10,090	448,318	19,090	118,000	3.61E+07	511,074	0.090
world	34,506	582,064	67,147	198,883	4.12E+08	2,149,318	0.430
mod2	34,774	604,910	66,409	199,810	4.07E+08	2,193,281	0.420
lp_maros_r7	3,136	664,080	9,408	144,848	7.35E+08	1,410,013	0.130
ex3sta1	17,443	679,857	17,516	68,779	7.73E+09	8,054,982	0.210
fxm3_16	41,340	765,526	85,575	392,252	2.84E+07	720,939	0.450
stat96v5	2,307	1,790,467	75,779	233,921	2.56E+09	2,172,256	0.210

TABLE 5.2
Hypergraph properties.

		$\mathcal{H}^2$		$\mathcal{H}^3$		$\mathcal{H}^4$		$\mathrm{H}_{\mathrm{RN}}(\tilde{A})$	
Name	*#nets*	*#nodes*	*#pins*	*#nodes*	*#pins*	*#nodes*	*#pins*	*#nodes*	*#pins*
General Matrices									
ncvxqp9	16,554	23,047	45,540	1.00	1.00	1.00	1.00	—	—
aug3dcqp	35,543	50,286	100,572	1.00	1.00	1.00	1.00	—	—
c-53	30,235	170,989	341,978	1.00	1.00	1.00	1.00	—	—
c-59	41,282	219,627	439,254	1.00	1.00	1.00	1.00	—	—
c-67	57,975	236,980	473,960	1.00	1.00	1.00	1.00	—	—
lshp3025	3,025	8,904	17,808	0.35	0.53	0.35	0.53	—	—
lshp3466	3,466	10,215	20,430	0.35	0.53	0.35	0.53	—	—
bodyy4	17,546	52,196	104,392	0.36	0.53	0.36	0.53	—	—
rail_20209	20,209	59,512	119,024	0.48	0.71	0.48	0.71	—	—
cvxbqp1	50,000	149,984	299,968	0.45	0.67	0.45	0.67	—	—
shuttle_eddy	10,429	46,585	93,170	0.51	0.75	0.36	0.53	—	—
nasa4704	4,704	50,026	100,052	0.48	0.72	0.30	0.60	—	—
bcsstk24	3,562	78,174	156,348	0.49	0.73	0.31	0.62	—	—
skirt	12,598	91,964	183,925	0.48	0.72	0.30	0.55	—	—
bcsstk28	4,410	107,307	214,614	0.49	0.73	0.31	0.63	—	—
s1rmq4m1	5,489	137,811	275,622	0.49	0.74	0.32	0.64	—	—
vibrobox	12,328	165,250	330,500	0.50	0.74	0.33	0.62	—	—
crystk01	4,875	155,508	311,016	0.50	0.74	0.32	0.63	—	—
bcsstm36	23,052	165,097	319,314	0.52	0.74	0.34	0.60	—	—
gridgena	48,962	231,561	463,122	0.54	0.74	0.39	0.59	—	—
k1_san	67,759	256,411	512,821	0.45	0.65	0.27	0.49	—	—
finan512	74,752	261,120	522,240	0.49	0.68	0.25	0.43	—	—
msc23052	23,052	565,881	1,131,762	0.49	0.74	0.32	0.64	—	—
bcsstk35	30,237	709,963	1,419,926	0.49	0.73	0.31	0.62	—	—
oilpan	73,752	1,761,718	3,523,436	0.49	0.74	0.30	0.60	—	—
geomean				**0.54**	**0.74**	**0.41**	**0.65**	—	—
LP Problems									
lp_pds_02	2953	10164	20328	0.75	0.83	0.74	0.81	0.74	0.81
delf	3170	15169	30338	0.48	0.70	0.34	0.60	0.18	0.31
lp_dfl001	6071	38098	76196	0.51	0.70	0.37	0.56	0.28	0.44
model9	2879	50730	101271	0.51	0.75	0.33	0.62	0.13	0.48
nl	7039	49034	98059	0.52	0.74	0.36	0.61	0.16	0.37
ge	10099	51015	102030	0.50	0.72	0.35	0.60	0.18	0.31
nemsemm2	6943	69269	138504	0.50	0.74	0.35	0.62	0.20	0.34
lp_nug12	3192	74592	149184	0.49	0.73	0.24	0.48	0.12	0.26
lp_ken_13	28632	66586	133172	0.62	0.72	0.62	0.72	0.64	0.73
lpi_gosh	3792	101213	202322	0.52	0.75	0.35	0.66	0.10	0.47
cq9	9278	106187	212343	0.50	0.74	0.34	0.60	0.11	0.33
lp_osa_14	2337	113843	227686	0.51	0.76	0.48	0.74	0.46	0.73
co9	10789	119330	238538	0.50	0.74	0.35	0.63	0.10	0.33
pltexpa	26894	121421	242842	0.51	0.69	0.43	0.63	0.33	0.46
model10	4400	144431	288861	0.51	0.75	0.33	0.62	0.10	0.49
fome12	24284	152392	304784	0.51	0.70	0.37	0.56	0.28	0.44
lp_cre_d	8926	184120	365790	0.57	0.78	0.47	0.68	0.38	0.58
r05	5190	200503	400987	0.50	0.74	0.34	0.66	0.04	0.26
p010	10090	219123	438237	0.49	0.74	0.35	0.66	0.08	0.27
world	34506	274179	547958	0.49	0.72	0.34	0.60	0.11	0.29
mod2	34774	285487	570555	0.49	0.72	0.34	0.60	0.10	0.28
lp_maros_r7	3136	330472	660944	0.50	0.75	0.35	0.70	0.01	0.11
ex3sta1	17443	331207	662414	0.49	0.73	0.31	0.61	0.02	0.08
fxm3_16	41340	362093	724186	0.51	0.74	0.37	0.65	0.13	0.29
stat96v5	2307	894082	1788162	0.50	0.75	0.34	0.68	0.00	0.01
geomean				*0.52*	*0.74*	*0.37*	*0.63*	**0.12**	**0.30**

footprint of the $\mathcal{H}^2$ hypergraph will be twice that of the graph representation of the respective matrix. However, as seen in the table, this size will be reduced by using $\mathcal{H}^4$ such that for many of the matrices, the memory footprint will be almost the same. For LP problems, the use of A matrix drastically reduces the memory footprint, and for the majority of the problems memory footprints of hypergraph-based ordering will be much smaller than those of a graph-based tool.

Tables 5.3 and 5.4 compare the ordering quality of the tools in terms of operation-count and fill-in metrics, respectively. In these two tables, ordering performances are displayed as normalized with respect to those of onmetis. In these two tables and the following tables and figures, COLAMD represents SYMAMD results on general matrices and COLAMD results on LP matrices.

First, we discuss the relative ordering quality performance of existing methods and tools on the results displayed in Tables 5.3 and 5.4. The onmetis is the clear winner on the average for ordering both general and LP matrices in terms of both operation-count and fill-in metrics. For the ordering of LP matrices, AMD, COLAMD and SMOOTH show close performances on the average. For the ordering of general matrices, AMD and COLAMD show better performance than SMOOTH on the average. Comparison of AMD and COLAMD for general matrices reveals that they display close performance in terms of fill-in metric, whereas AMD shows better performance than COLAMD in terms of operation-count metric, on the average. As seen in Table 5.5, AMD is the fastest for both general and LP matrices, whereas COLAMD is the second fastest in both general and LP matrices, on the average.

Second, we discuss the effect of different clique cover finding algorithms and ordering schemes implemented in oPaToH. As seen in Tables 5.3 and 5.4, the ordering quality of oPaToH increases in general when larger cliques of $\mathrm{G}(M)$ are considered while constructing the hypergraph. That is, in general, oPaToH using $\mathcal{H}^4$ produces better orderings than oPaToH using $\mathcal{H}^3$, which in turn produces better orderings than oPaToH using $\mathcal{H}^2$. For LP matrices, oPaToH using $\mathrm{H}_{\mathrm{RN}}(\tilde{A})$ usually produces better orderings than oPaToH using $\mathcal{H}^2$, $\mathcal{H}^3$, and $\mathcal{H}^4$. These results justify our earlier choice on the use of $\mathrm{H}_{\mathrm{RN}}(\tilde{A})$ for ordering LP matrices.

Third, we discuss the ordering performance of oPaToH with respect to onmetis, since onmetis appears to be the best existing ordering tool, on the overall average. As seen in Tables 5.3 and 5.4, oPaToH produces considerably better orderings than onmetis, for both general and LP matrices, where the performance gap is more pronounced in the ordering of LP matrices. As seen in Table 5.3, the ordering quality of oPaToH increases with increasing clique sizes used in clique-node hypergraph construction, on the average. For example, for general matrices, oPaToH using $\mathcal{H}^2$, $\mathcal{H}^3$, and $\mathcal{H}^4$ produce orderings with 10%, 13%, and 14% less operation count than onmetis, respectively, on the average. For LP matrices, oPaToH using $\mathrm{H}_{\mathrm{RN}}(\tilde{A})$ produces orderings with 20% less operation count than onmetis, on the average. Comparison of Tables 5.3 and 5.4 shows that the performance gap between oPaToH and onmetis is smaller in terms of fill-in metric than in terms of operation-count metric, as expected. As seen in Table 5.4, for general matrices, oPaToH using $\mathcal{H}^2$, $\mathcal{H}^3$, and $\mathcal{H}^4$ produce orderings with 6%, 7%, and 8% less nonzeros in factor matrices than onmetis, respectively, on the average. For LP matrices, oPaToH using $\mathrm{H}_{\mathrm{RN}}(\tilde{A})$ produces orderings with 10% less nonzeros in factor matrices than onmetis, on the average. Since oPaToH and onmetis are HP-based and GPVS-based ordering tools, respectively, the better quality orderings produced by oPaToH confirm the validity of our HP-based GPVS formulation in the application of fill-reducing ordering of sparse matrices.

TABLE 5.3
Operation counts of various ordering methods and tools relative to onmetis.

Name	AMD	COLAMD	SMOOTH	oPaToH $\mathcal{H}^2$	oPaToH $\mathcal{H}^3$	oPaToH $\mathcal{H}^4$	oPaToH $\mathrm{H}_{\mathrm{RN}}(\tilde{A})$
General Matrices							
ncvxqp9	1.12	1.73	1.56	0.65	0.65	0.65	—
aug3dcqp	1.03	1.06	1.56	0.72	0.72	0.72	—
c-53	2.55	0.95	1.53	0.72	0.72	0.72	—
c-59	1.18	0.96	1.22	1.09	1.09	1.09	—
c-67	1.96	0.88	0.89	0.94	0.94	0.94	—
lshp3025	0.98	0.91	1.07	0.92	0.91	0.91	—
lshp3466	1.09	0.95	1.14	0.93	0.90	0.90	—
bodyy4	1.04	1.45	1.56	0.98	0.98	0.98	—
rail_20209	1.12	1.07	1.16	1.08	1.02	1.02	—
cvxbqp1	0.91	5.72	5.80	0.84	0.84	0.86	—
shuttle_eddy	0.98	0.84	1.18	1.01	0.97	0.95	—
nasa4704	0.83	1.01	0.83	0.69	0.67	0.68	—
bcsstk24	1.14	0.85	0.89	0.91	0.88	0.96	—
skirt	0.97	1.13	1.14	1.08	1.00	1.01	—
bcsstk28	0.78	0.69	0.67	0.79	0.78	0.76	—
s1rmq4m1	0.80	0.97	1.05	0.88	0.91	0.95	—
vibrobox	1.04	1.09	0.87	1.09	0.85	0.68	—
crystk01	0.78	1.16	1.27	1.08	1.07	1.13	—
bcsstm36	0.97	0.90	0.91	0.81	0.81	0.80	—
gridgena	1.05	1.04	1.16	1.05	0.97	0.98	—
k1_san	0.96	3.44	3.02	1.17	0.81	0.88	—
finan512	1.08	1.64	20.68	0.75	0.82	0.62	—
msc23052	1.11	0.97	0.95	0.87	0.88	0.88	—
bcsstk35	1.04	0.75	0.78	0.82	0.82	0.80	—
oilpan	1.06	1.47	1.41	0.98	1.00	1.04	—
geomean	**1.06**	**1.16**	**1.37**	**0.90**	**0.87**	**0.86**	
LP Problems							
lp_pds_02	1.18	1.17	0.90	1.66	0.83	0.84	0.89
delf	0.92	0.86	0.94	0.90	0.86	0.82	0.84
lp_dfl001	1.77	1.74	1.79	0.69	0.66	0.65	0.65
model9	0.72	0.89	0.81	0.66	0.65	0.65	0.64
nl	0.89	0.88	0.91	0.93	0.94	0.94	0.96
ge	1.59	1.70	1.42	1.14	0.99	0.98	0.82
nemsemm2	0.76	0.96	0.80	0.84	0.80	0.79	0.78
lp_nug12	1.20	1.16	1.26	1.11	1.24	1.21	0.89
lp_ken_13	0.89	0.92	0.94	0.92	0.92	0.93	0.92
lpi_gosh	1.04	1.17	1.06	0.87	0.99	0.76	0.70
cq9	1.23	1.25	1.32	1.00	1.02	0.98	0.97
lp_osa_14	1.00	1.00	1.00	1.00	1.00	1.00	1.00
co9	1.10	1.06	1.16	0.95	0.94	0.93	0.87
pltexpa	2.11	0.99	6.16	1.01	0.69	0.67	0.71
model10	1.72	3.79	1.92	1.06	0.93	0.94	0.92
fome12	1.76	1.76	1.81	0.71	0.66	0.65	0.66
lp_cre_d	1.43	1.06	1.46	1.23	1.00	0.97	0.85
r05	0.75	0.53	0.60	0.43	0.44	0.44	0.43
p010	1.39	1.12	1.20	0.91	0.91	0.92	0.91
world	0.64	0.64	0.67	0.75	0.73	0.73	0.65
mod2	0.67	0.64	0.67	0.76	0.75	0.73	0.68
lp_maros_r7	0.67	0.33	0.66	1.19	1.03	0.98	0.92
ex3sta1	8.03	8.69	9.53	1.52	1.14	1.34	1.00
fxm3_16	0.71	1.07	0.69	1.06	0.93	0.84	0.93
stat96v5	1.60	1.60	0.76	0.94	0.89	0.87	0.78
geomean	**1.17**	**1.15**	**1.19**	*0.94*	*0.86*	*0.84*	**0.80**

TABLE 5.4
Factor nonzero counts of various ordering methods and tools relative to onmetis.

Name	AMD	COLAMD	SMOOTH	oPaToH $\mathcal{H}^2$	oPaToH $\mathcal{H}^3$	oPaToH $\mathcal{H}^4$	oPaToH $\mathrm{H}_{\mathrm{RN}}(\tilde{A})$
General Matrices							
ncvxqp9	1.02	1.03	1.01	0.83	0.83	0.83	—
aug3dcqp	1.03	0.88	1.07	0.78	0.78	0.78	—
c-53	1.36	0.92	1.28	0.96	0.96	0.96	—
c-59	1.07	0.89	1.00	0.96	0.96	0.96	—
c-67	1.16	0.92	0.93	0.93	0.93	0.93	—
lshp3025	0.99	0.95	1.00	0.96	0.95	0.95	—
lshp3466	1.03	0.96	1.03	0.96	0.96	0.96	—
bodyy4	1.01	1.09	1.13	0.97	0.97	0.97	—
rail_20209	1.04	0.99	1.02	1.01	0.98	0.98	—
cvxbqp1	0.96	1.91	1.89	0.92	0.92	0.92	—
shuttle_eddy	1.00	0.92	1.08	0.99	0.97	0.97	—
nasa4704	0.96	0.96	0.89	0.84	0.83	0.84	—
bcsstk24	1.07	0.91	0.93	0.95	0.93	0.96	—
skirt	0.98	1.01	1.01	1.00	0.98	0.98	—
bcsstk28	0.95	0.87	0.85	0.91	0.91	0.90	—
s1rmq4m1	0.92	0.97	1.01	0.95	0.96	0.97	—
vibrobox	1.01	1.00	0.94	0.96	0.88	0.80	—
crystk01	0.88	1.03	1.09	1.02	1.02	1.04	—
bcsstm36	0.97	0.93	0.93	0.89	0.90	0.89	—
gridgena	1.01	1.02	1.08	1.00	0.97	0.97	—
k1_san	0.99	1.63	1.60	1.07	0.91	0.94	—
finan512	1.05	1.13	2.79	0.91	0.91	0.85	—
msc23052	1.03	0.93	0.93	0.92	0.92	0.92	—
bcsstk35	1.02	0.88	0.89	0.91	0.91	0.90	—
oilpan	1.04	1.10	1.08	0.96	0.97	0.99	—
geomean	**1.02**	**1.01**	**1.09**	**0.94**	**0.93**	**0.92**	
LP Problems							
lp_pds_02	0.99	0.99	0.91	1.09	0.90	0.90	0.92
delf	0.93	0.92	0.94	0.93	0.92	0.91	0.91
lp_dfl001	1.25	1.23	1.26	0.84	0.82	0.81	0.81
model9	0.89	0.98	0.93	0.85	0.84	0.84	0.84
nl	0.93	0.93	0.94	0.94	0.94	0.94	0.95
ge	1.08	1.10	1.06	0.99	0.95	0.95	0.91
nemsemm2	0.89	0.98	0.90	0.91	0.90	0.90	0.90
lp_nug12	1.08	1.06	1.10	1.10	1.17	1.14	0.95
lp_ken_13	0.92	0.93	0.94	0.95	0.95	0.95	0.95
lpi_gosh	0.97	1.06	0.97	0.93	0.97	0.89	0.87
cq9	1.06	1.04	1.07	0.98	0.98	0.97	0.96
lp_osa_14	1.00	1.00	1.00	1.00	1.00	1.00	1.00
co9	0.99	0.98	1.01	0.94	0.94	0.94	0.92
pltexpa	1.16	0.99	1.82	0.95	0.80	0.80	0.79
model10	1.23	1.74	1.30	1.03	0.98	0.99	0.96
fome12	1.25	1.24	1.26	0.84	0.81	0.81	0.81
lp_cre_d	1.14	1.00	1.15	1.08	0.98	0.98	0.93
r05	1.16	0.86	0.91	0.81	0.81	0.81	0.81
p010	1.26	0.99	1.02	0.94	0.94	0.95	0.94
world	0.81	0.82	0.83	0.86	0.85	0.85	0.81
mod2	0.83	0.82	0.83	0.87	0.86	0.86	0.83
lp_maros_r7	0.84	0.61	0.83	1.07	1.01	0.98	0.95
ex3sta1	3.03	3.07	3.19	1.17	1.03	1.13	0.96
fxm3_16	0.89	1.10	0.88	0.98	0.94	0.92	0.95
stat96v5	1.22	1.22	0.88	0.97	0.95	0.94	0.90
geomean	**1.07**	**1.05**	**1.06**	*0.96*	*0.93*	*0.92*	**0.90**

TABLE 5.5
Total execution times of various ordering methods and tools relative to onmetis.

Name	AMD	COLAMD	SMOOTH	oPaToH $\mathcal{H}^2$	oPaToH $\mathcal{H}^3$	oPaToH $\mathcal{H}^4$	oPaToH $\mathrm{H}_{\mathrm{RN}}(\tilde{A})$
General Matrices							
ncvxqp9	0.16	0.17	2.25	2.84	2.86	2.88	—
aug3dcqp	0.10	0.11	3.54	2.55	2.57	2.58	—
c-53	0.40	1.28	14.26	15.56	15.64	15.72	—
c-59	0.16	0.37	13.24	7.21	7.35	7.54	—
c-67	0.30	1.00	4.17	8.20	8.25	8.31	—
lshp3025	0.24	0.29	2.19	6.93	4.45	4.53	—
lshp3466	0.28	0.33	2.60	7.99	5.25	5.35	—
bodyy4	0.20	0.22	2.04	5.88	3.59	3.65	—
rail_20209	0.16	0.20	1.76	4.58	3.13	3.17	—
cvxbqp1	0.25	0.28	2.30	6.50	4.46	4.55	—
shuttle_eddy	0.17	0.23	2.03	6.99	4.79	4.31	—
nasa4704	0.26	0.49	3.76	4.30	4.38	4.74	—
bcsstk24	0.42	1.11	8.61	4.70	6.69	9.73	—
skirt	0.19	0.29	2.05	8.38	6.00	5.12	—
bcsstk28	0.55	1.53	11.39	4.29	6.56	11.17	—
s1rmq4m1	0.69	1.84	16.66	5.22	8.20	14.14	—
vibrobox	0.22	0.41	5.13	11.72	8.30	7.02	—
crystk01	0.45	1.26	13.38	7.92	9.93	15.61	—
bcsstm36	0.06	0.13	5.16	0.89	0.96	1.14	—
gridgena	0.18	0.24	1.99	6.89	4.87	4.15	—
k1_san	0.10	0.15	1.85	4.91	2.96	2.33	—
finan512	0.14	0.20	2.56	5.74	3.66	2.37	—
msc23052	0.72	2.47	17.67	5.73	8.35	13.43	—
bcsstk35	0.45	1.39	10.88	5.00	6.61	9.71	—
oilpan	0.77	2.22	23.38	4.92	7.80	13.83	—
geomean	**0.25**	**0.46**	**4.81**	**5.54**	**5.16**	**5.70**	—
LP Problems							
lp_pds_02	0.27	0.29	2.26	3.52	3.20	3.24	3.09
delf	0.18	0.16	1.76	4.49	3.38	3.04	1.77
lp_dfl001	0.66	0.64	10.08	7.96	6.78	6.25	5.14
model9	0.18	0.23	1.78	5.71	6.23	5.67	2.61
nl	0.52	0.60	9.84	9.78	7.31	7.03	3.77
ge	0.19	0.16	2.22	4.10	3.18	2.87	1.90
nemsemm2	0.25	0.53	1.92	8.46	7.50	5.99	2.74
lp_nug12	0.41	0.49	12.87	9.41	6.11	5.12	2.95
lp_ken_13	0.52	0.49	3.29	8.77	6.17	6.66	6.08
lpi_gosh	0.57	0.73	11.60	17.12	13.69	12.92	5.88
cq9	0.89	1.53	12.92	18.69	14.17	14.09	5.76
lp_osa_14	0.13	0.19	1.43	1.96	13.47	38.45	2.67
co9	1.01	1.87	15.72	22.25	17.18	16.76	7.28
pltexpa	0.17	0.14	2.01	4.20	3.22	2.92	2.34
model10	0.32	0.45	10.08	14.03	14.91	13.31	5.43
fome12	0.53	0.50	7.99	9.44	6.92	6.07	4.97
lp_cre_d	0.54	1.24	23.45	31.43	21.31	17.35	12.39
r05	0.27	0.29	3.86	7.01	5.70	6.22	2.73
p010	0.32	0.74	4.33	7.50	5.09	6.11	2.66
world	0.46	0.48	6.16	10.38	7.25	5.94	3.00
mod2	0.46	0.45	6.25	10.42	7.41	6.04	2.94
lp_maros_r7	0.24	2.17	7.11	64.19	36.66	37.44	2.75
ex3sta1	0.34	0.16	26.21	8.84	8.48	7.66	1.99
fxm3_16	0.14	0.14	1.56	6.94	5.81	5.25	1.79
stat96v5	0.56	0.07	10.79	251.54	149.93	129.38	2.43
geomean	**0.35**	**0.41**	**5.50**	*10.42*	*8.79*	*8.52*	**3.41**

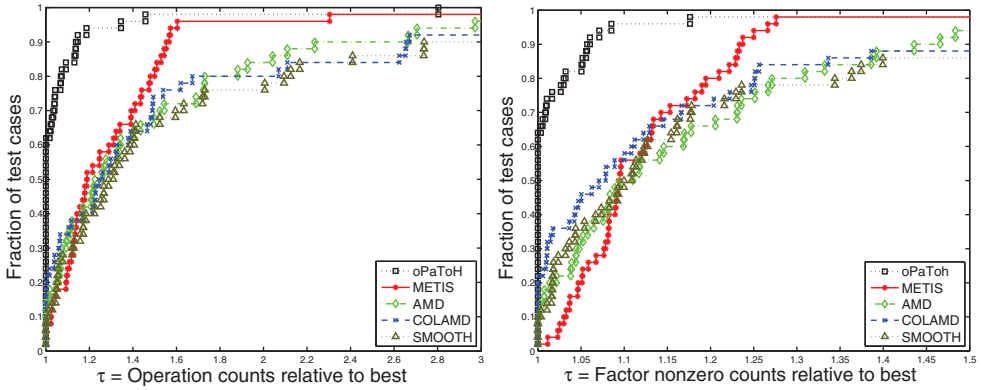


FIG. 5.1. *Performance profile charts for operation counts and factor nonzero counts.*

For a visual summary of the results we presented in Tables 5.3 and 5.4, we use *performance profiles* [25], a visual tool for comparing various *methods* over a large set of *test cases* with respect to the same *metric*. The metrics we use in the following profile plots are the operation counts and factor nonzero counts. In Figure 5.1, each performance profile plot shows the probability that a specific method gives results within some value τ multiple of the best result reached by all of the methods. The higher the probability of small τ values, the more preferable the method is. For example, in Figure 5.1(b), the curve oPaToH roughly passes through the point (1.1, 0.9), which means that on 90% of the tested cases, the factor nonzero count found by oPaToH was no more than 1.1 times more than the best obtained by any method. In other words, a curve closer to the left means the method's solution is closer to best known solutions than the others. These charts show that oPaToH produces more favorable orderings than all four other methods compared in this experiment.

Table 5.5 displays total execution times of ordering methods and tools as normalized with respect to those of onmetis. As seen in the table, AMD is the fastest ordering method and COLAMD is the second fastest. For general matrices, searching for 3-cliques in the construction of clique-node hypergraphs amortizes its cost in 11 out of 20 matrices by reducing the total ordering time. oPaToH using $\mathcal{H}^3$ takes 6.9% less ordering time than oPaToH using $\mathcal{H}^2$, on the average. However, as seen in the table, searching for 4-cliques in the construction of clique-node hypergraphs amortizes its cost in only 6 out of 15 matrices. oPaToH using $\mathcal{H}^4$ takes 2.8% more ordering time than oPaToH using $\mathcal{H}^2$, on the average. As seen in Table 5.5, oPaToH is significantly slower than onmetis for the ordering of general matrices. However, for LP matrices, oPaToH using $\mathrm{H_{RN}}(\tilde{A})$ is quite fast, and it is 241% slower than onmetis, on the average. The slower run-time performance of oPaToH compared to onmetis is expected, because hypergraph partitioning is computationally more expensive than graph partitioning, in general.

Table 5.6 displays results about the structure of the elimination tree, and solver performance results for onmetis, AMD, and oPaToH. We report average leaf depths of the elimination trees produced by AMD and oPaToH as normalized with respect to those by onmetis, whereas average leaf depths produced by onmetis are given in absolute values. The standard deviations of the leaf depths are given in absolute values for all tools. The structure of the elimination tree is a relevant metric for the ability of the linear system to be solved in parallel since broader and shorter trees are

TABLE 5.6
Elimination tree and solver performances.

	Elimination tree statistics (leaf depths)						Solver statistics					
	onmetis		AMD		oPaToH		onmetis		AMD		oPaToH	
Name	*avg*	*std*	*avg*	*std*	*avg*	*std*	time	opc	time	opc	time	opc
General Matrices												
ncvxqp9	169	50.4	1.08	53.9	0.92	48.1	0.050	1.28E+07	1.01	1.63	0.82	0.65
aug3dcqp	790	111.8	1.01	109.5	0.91	120.4	0.620	5.77E+08	0.76	1.05	0.59	0.72
c-53	331	72.8	1.65	113.7	0.75	49.9	21.417	9.17E+09	0.11	0.18	0.22	0.27
c-59	1306	514.7	1.05	528.1	0.99	710.3	62.949	4.56E+10	0.19	0.38	0.38	0.34
c-67	122	58.1	1.96	98.7	1.15	64.7	0.182	2.59E+07	0.82	0.84	0.90	0.94
lshp3025	163	10.8	1.02	9.9	0.98	24.0	0.090	6.49E+07	1.61	2.19	0.79	0.77
lshp3466	178	9.7	1.03	10.2	0.96	24.5	0.113	8.23E+07	1.24	1.66	0.86	0.88
bodyy4	363	12.5	1.02	10.3	1.00	38.2	0.132	6.93E+07	1.13	1.58	0.92	0.98
rail_20209	230	36.1	1.05	40.6	0.98	69.3	0.090	2.85E+07	0.94	1.01	0.93	1.02
cvxbqp1	837	307.6	0.97	302.9	1.01	369.4	1.064	9.90E+08	4.33	6.81	0.92	0.85
shuttle_eddy	266	16.6	0.98	14.4	0.98	27.2	0.655	5.76E+08	9.51	12.80	0.84	0.86
nasa4704	405	23.7	0.94	20.3	0.87	57.3	0.078	7.78E+07	0.92	0.94	0.76	0.68
bcsstk24	443	60.9	0.97	25.5	0.91	38.3	0.079	8.43E+07	0.79	0.78	0.93	0.96
skirt	167	76.3	0.96	73.3	0.92	83.9	0.603	5.89E+08	4.89	5.96	0.93	0.92
bcsstk28	424	77.5	0.87	24.4	0.86	64.7	0.100	1.11E+08	0.74	0.64	0.83	0.76
s1rmq4m1	521	37.4	0.96	8.4	1.02	72.7	0.173	2.19E+08	0.97	1.00	0.95	0.95
vibrobox	1247	66.6	1.01	53.3	0.91	188.6	1.331	2.02E+09	0.87	0.92	0.66	0.68
crystk01	676	29.9	0.96	23.8	1.08	67.9	1.587	2.56E+09	2.19	2.29	0.86	0.80
bcsstm36	37	138.3	0.88	127.9	0.90	126.7	0.235	2.35E+08	0.89	0.90	0.83	0.80
gridgena	713	29.0	1.03	73.9	0.98	100.9	0.765	7.25E+08	0.96	1.03	0.92	0.98
k1_san	822	48.3	0.97	54.7	0.94	124.7	0.833	8.30E+08	1.75	2.42	0.87	0.88
finan512	349	55.3	1.06	58.9	0.84	32.6	0.506	3.07E+08	2.14	4.13	0.89	0.62
msc23052	917	91.6	1.05	80.5	0.93	248.1	0.980	1.30E+09	0.91	0.93	0.87	0.88
bcsstk35	749	142.3	1.01	133.2	0.89	141.4	0.867	1.03E+09	0.79	0.75	0.84	0.80
oilpan	1474	92.3	1.01	44.0	1.06	140.8	3.804	5.64E+09	1.37	1.49	1.01	1.04
geomean			**1.04**		**0.95**				**1.12**	**1.38**	**0.87**	**0.83**
LP Problems												
lp_pds_02	130	33.6	1.35	52.7	0.78	27.2	0.013	3.88E+06	0.91	1.17	0.92	0.89
delf	110	39.6	1.31	48.7	1.01	35.4	0.014	3.89E+06	0.94	0.92	0.88	0.84
lp_dfl001	1064	119.6	1.43	229.9	0.76	144.5	1.251	1.45E+09	1.46	1.77	0.86	0.66
model9	76	44.3	1.75	70.7	0.82	34.2	0.024	1.08E+07	0.85	0.72	0.85	0.65
nl	335	66.1	1.03	96.7	0.91	105.7	0.117	8.42E+07	0.85	0.89	0.99	0.96
ge	329	48.3	1.32	79.5	0.87	71.6	0.087	4.95E+07	1.11	1.58	0.89	0.82
nemsemm2	68	43.4	2.97	58.2	0.79	36.7	0.041	1.28E+07	0.86	0.76	0.93	0.79
lp_nug12	1758	391.9	1.29	18.2	1.05	119.0	4.196	6.71E+09	1.04	1.20	0.87	0.90
lp_ken_13	90	6.3	0.99	5.7	0.99	5.8	0.146	4.07E+07	0.98	0.81	1.07	0.98
lpi_gosh	309	119.3	1.28	136.8	0.84	98.9	0.103	7.99E+07	0.86	1.04	0.86	0.70
cq9	267	67.0	1.29	87.8	0.98	84.3	0.149	8.60E+07	0.96	1.23	0.98	0.97
lp_osa_14	74	0.5	1.00	1.8	0.93	3.8	0.047	1.32E+07	0.97	1.00	0.96	1.00
co9	304	81.8	1.19	108.0	0.90	92.3	0.188	1.12E+08	0.88	1.10	0.94	0.87
pltexpa	474	116.9	2.44	181.1	0.83	100.1	0.368	3.77E+08	1.59	2.11	0.81	0.71
model10	383	85.5	1.77	193.9	0.90	79.6	0.126	1.15E+08	1.26	1.72	1.02	0.92
fome12	1054	123.7	1.41	286.3	0.76	146.2	4.861	5.72E+09	1.46	1.77	0.90	0.66
lp_cre_d	325	326.3	1.31	425.1	0.83	291.7	0.383	4.20E+08	1.19	1.43	0.87	0.85
r05	599	91.9	0.49	62.1	0.40	41.9	0.242	2.45E+08	1.45	0.75	0.64	0.43
p010	166	25.3	1.28	30.0	0.91	23.0	0.195	7.28E+07	1.38	1.39	0.74	0.91
world	430	154.1	1.74	218.2	0.80	115.7	0.869	8.26E+08	0.69	0.64	0.79	0.65
mod2	440	148.3	1.94	293.7	0.75	111.7	0.901	8.15E+08	0.68	0.67	0.77	0.68
lp_maros_r7	931	68.5	1.26	371.9	0.99	93.9	0.860	1.47E+09	0.72	0.67	1.02	0.92
ex3sta1	2318	82.2	2.33	836.4	1.04	425.3	9.875	1.55E+10	6.75	8.02	0.95	1.00
fxm3_16	211	35.2	1.30	35.9	0.89	56.8	0.197	5.75E+07	0.88	0.71	1.04	0.93
stat96v5	997	889.2	0.77	1086.1	1.02	834.4	4.538	5.11E+09	0.89	1.60	0.84	0.78
geomean			**1.36**		**0.86**				**1.09**	**1.17**	**0.89**	**0.80**

more amenable to parallelization. As seen in the table, oPaToH leads to noticeably favorable elimination trees, in particular in LP problems.

As a solver, we used SuperLU [24] in "Symmetric Mode."[3] Like the other state-of-the-art solvers, SuperLU is designed to use high performance Basic Linear Algebra Subroutines (BLAS) libraries. We used SuperLU with vendor optimized AMD Core Math Library v4.4.0. Since it uses BLAS, actual operation counts during the solution can be higher than the ones we reported earlier. Hence, in addition to solver time (in seconds) this table also includes the solver operation counts (abbreviated as "opc"). In the table, onmetis results are given as absolute values, and again, AMD and oPaToH results are normalized with respect to onmetis results. As seen in the table, orderings obtained by oPaToH lead to significantly faster solution times and smaller solver operation counts compared to those by onmetis and AMD. These results show the merits of the proposed HP-based ordering method in practice.

The above discussions given on Tables 5.3–5.5 show that oPaToH produces considerably better quality orderings than onmetis at a higher computational cost. Thus, the higher computational cost of oPaToH can be typically justified for applications that involve multiple numerical factorization of matrices with the same sparsity patterns and/or multiple solutions with different right-hand side vectors. Interior point methods that adopt the normal equations approach constitute such a typical case. This is because the numerical factorization $M = LL^T$ of matrix M is required at each iteration, where the sparsity pattern of matrix $M = AD^2A^T$ is independent of the value of the diagonal D^2 matrix and hence remains the same at all iterations.

6. Conclusion and future work. Direct solvers are one of the preferred methods for solving linear systems due to their numerical robustness. A typical first step in this process is reordering of the input matrix to improve execution time and space requirements of the solution process. Graphs have been extensively used to model the evolution of the nonzero structure during the factorization step of direct solvers and hence for the reordering process. Decades after the first theoretical work on nested dissection, advances in multilevel graph partitioning finally enabled the development of long-awaited, successful nested dissection-based ordering tools that work for a wider range of problems. The state-of-the-art nested dissection-based ordering tools directly employ graph partitioning by vertex separator (GPVS). In this work, we presented that GPVS has a deficiency in the multilevel framework, where a vertex separator found in the coarser levels may not be a narrow separator in the original graph. We introduced a novel hypergraph partitioning (HP) formulation of GPVS that is not vulnerable to GPVS's deficiency in the multilevel framework. We developed a novel HP-based fill-reducing ordering method. In matrix terms, our approach relies on the existence of a structural factorization of a symmetric matrix M in the form of $M = AA^T$, where A is a rectangular matrix. Such structural factorizations arise in different contexts, such as solution of LP problems, where $M = AD^2A^T$ and D^2 is a diagonal matrix. In the absence of such structural factorization, we also proposed simple, yet effective, structural factorization techniques that can be applied to any arbitrary symmetric matrix. For matrices coming from LP problems, we also proposed two structural factor sparsening methods. We developed an HP-based fill-reducing ordering tool oPaToH by embedding the proposed HP-based GPVS formulation and suggested methods into the state-of-the-art HP tool PaToH.

[3]We use default options and set `DiagPivotThresh=0.0` to make all diagonal entries to be an acceptable pivot.

We performed our experimental evaluations using 50 publicly available test matrices, where 25 of them come from LP problems and 25 are general symmetric matrices. We compared the performance of oPaToH against the state-of-the-art ordering tools onmetis [44], AMD [2], COLAMD [23], and SMOOTH [6]. Among the existing tools we tested, in general, GPVS-based onmetis produces best results in terms of operation counts and amount of fill-in. In terms of operation counts, oPaToH produced orderings that require 14% and 20% less operation counts than the ones produced by onmetis for general and LP matrices, respectively, on the average. In terms of number of nonzero counts in the triangular factors, oPaToH produced 7% and 10% less nonzeros in comparison to onmetis for general and LP matrices, respectively. These reductions come at the expense of higher execution time, which can easily be amortized in applications involving multiple numerical factorization of matrices with the same sparsity patterns and/or multiple solutions with different right-hand side vectors.

As a future work, although we have shown that higher ordering cost can easily be amortized, we will continue to investigate to improve the runtime performance of our tool. We observed that one of the reasons for slower execution time in our ordering tool is due to MD codes we used [6]. We are planning to investigate faster alternatives, such as constrainted approximate minimum degree [3].

Nested dissection is known to be asymptotically optimum for certain classes of problems [30].[4] It would be interesting to investigate how graph- or hypergraph-based nested dissection tools behave. However, we feel such a study and also a theoretical investigation like [32] is beyond the scope of this paper and can remain as future work.

Acknowledgments. We thank Bora Uçar for his comments on an earlier version of the paper. We also would like to thank the anonymous referees for their valuable comments, which helped us improve the presentation of this paper.

REFERENCES

[1] C. J. Alpert and A. B. Kahng, *Recent directions in netlist partitioning: A survey*, VLSI J., 19 (1995), pp. 1–81.

[2] P. R. Amestoy, T. A. Davis, and I. S. Duff, *An approximate minimum degree ordering algorithm*, SIAM J. Matrix Anal. Appl., 17 (1996), pp. 886–905.

[3] P. R. Amestoy, T. A. Davis, and I. S. Duff, *Algorithm* 837: *AMD, an approximate minimum degree ordering algorithm*, ACM Trans. Math. Software, 30 (2004), pp. 381–388.

[4] E. D. Andersen, J. Gondzio, C. Meszaros, and X. Xu, *Implementation of interior point methods for large scale linear programming*, in Interior Point Methods in Mathematical Programming, Kluwer Academic Publishers, Norwell, MA, 1996, pp. 189–252.

[5] C. Ashcraft and R. Grimes, *Spooles: An object-oriented sparse matrix library*, in Proceedings of the 9th SIAM Conference on Parallel Processing for Scientific Computing, 1999.

[6] C. Ashcraft and J. W. H. Liu, *SMOOTH: A software package for ordering sparse matrices*, 1996.

[7] C. Ashcraft and J. W. H. Liu, *Applications of the Dulmage–Mendelsohn decomposition and network flow to graph bisection improvement*, SIAM J. Matrix Anal. Appl., 19 (1998), pp. 325–354.

[8] C. Aykanat, A. Pinar, and U. V. Çatalyürek, *Permuting sparse rectangular matrices into block-diagonal form*, SIAM J. Sci. Comput., 25 (2004), pp. 1860–1879.

[9] A. Brandstadt, V. B. Le, and J. P. Spinrad, *Graph classes: A survey*, SIAM Monogr. Discrete Math. Appl., SIAM, Philadelphia, 1999.

[10] T. N. Bui and C. Jones, *A heuristic for reducing fill-in in sparse matrix factorization*, in Proceedings of the 6th SIAM Conference on Parallel Processing for Scientific Computing, 1993, pp. 445–452.

[4] We refer readers to Section 3.1 of [26] for a succinct survey of ordering methods.

[11] U. V. Çatalyürek, *Hypergraph models for sparse matrix partitioning and reordering*, Ph.D. thesis, Bilkent University, Computer Engineering and Information Science, Bilkent, Turkey, 1999; available online from http://www.cs.bilkent.edu.tr/tech-reports/1999/ABSTRACTS.1999.html.

[12] U. V. Çatalyürek and C. Aykanat, *Hypergraph-partitioning based decomposition for parallel sparse-matrix vector multiplication*, IEEE Trans. Parallel Distributed Systems, 10 (1999), pp. 673–693.

[13] U. V. Çatalyürek and C. Aykanat, *PaToH: A Multilevel Hypergraph Partitioning Tool, Version* 3.0, Department of Computer Engineering, Bilkent University, Ankara, 06533 Turkey. PaToH is available from http://bmi.osu.edu/ umit/software.htm, 1999.

[14] U. V. Çatalyürek and C. Aykanat, *A fine-grain hypergraph model for* 2*D decomposition of sparse matrices*, in Proceedings of 15th International Parallel and Distributed Processing Symposium (IPDPS), San Francisco, CA, 2001.

[15] U. V. Çatalyürek and C. Aykanat, *Hypergraph-partitioning-based sparse matrix ordering*, in Proceedings of the Second International Workshop on Combinatorial Scientific Computing (CSC05), CERFACS, Toulouse, France, 2005.

[16] U. V. Çatalyürek, C. Aykanat, and E. Kayaaslan, *Hypergraph partitioning-based fill-reducing ordering*, Technical reports OSUBMI-TR-2009-n02 and BU-CE-0904, Department of Biomedical Informatics and Bilkent University, Computer Engineering Department, The Ohio State University, Columbus, OH, 2009.

[17] U. V. Çatalyürek, C. Aykanat, and B. Uçar, *On two-dimensional sparse matrix partitioning: Models, methods, and a recipe*, SIAM J. Sci. Comput., 32 (2010), pp. 656–683.

[18] C. Chevalier and F. Pellegrini, *PT-SCOTCH: A tool for efficient parallel graph ordering*, Parallel Comput., 34 (2008), pp. 318–331.

[19] J. Cong, W. Labio, and N. Shivakumar, *Multi-way VLSI circuit partitioning based on dual net representation*, in Proceedings of the IEEE International Conference on Computer-Aided Design, 1994, pp. 56–62.

[20] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, MIT Press, McGraw-Hill, New York, 1990.

[21] G. B. Dantzig and M. N. Thapa, *Linear Programming: Theory and Extensions*, Springer, New York, 2003.

[22] T. Davis, *University of Florida sparse matrix collection: http://www.cise.ufl.edu/research/sparse/*, NA Digest, 92/96/97 (1994/1996/1997).

[23] T. A. Davis, J. R. Gilbert, S. I. Larimore, and E. G. Ng, *A column approximate minimum degree ordering algorithm*, ACM Trans. Math. Software, 30 (2004), pp. 353–376.

[24] J. Demmel, S. Eisenstat, J. Gilbert, X. Li, and J. Liu, *A supernodal approach to sparse partial pivoting*, SIAM J. Matrix Anal. Appl., 20 (1999), pp. 720–755.

[25] E. D. Dolan and J. J. Moré, *Benchmarking optimization software with performance profiles*, Math. Program., 91 (2002), pp. 201–213.

[26] I. S. Duff and B. Uçar, *Combinatorial problems in solving linear systems*, Technical report TR/PA/09/60, CERFACS, Toulouse, France, 2009; available online from http://www.cerfacs.fr/algor/reports/.

[27] C. M. Fiduccia and R. M. Mattheyses, *A linear-time heuristic for improving network partitions*, in Proceedings of the 19th ACM/IEEE Design Automation Conference, 1982, pp. 175–181.

[28] A. George, M. Heath, and E. Ng, *A comparison of some methods for solving sparse linear least-squares problems*, SIAM J. Sci. Stat. Comput., 4 (1983), pp. 177–187.

[29] A. George and J. W. H. Liu, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[30] A. George, *Nested dissection of a regular finite element mesh*, SIAM J. Numer. Anal., 10 (1973), pp. 345–363.

[31] J. Gilbert and E. Ng, *Predicting structure in nonsymmetric sparse matrix factorizations*, IMA Vol. Math. Appl., 56 (1993), pp. 107–139.

[32] J. R. Gilbert and R. E. Tarjan, *The analysis of a nested dissection algorithm*, Numer. Math., 50 (1987), pp. 377–404.

[33] J. Gramm, J. Guo, F. Hüffner, and R. Niedermeier, *Data reduction, exact, and heuristic algorithms for clique cover*, in Proceedings of the Eighth Workshop on Algorithm Engineering, 2006.

[34] L. Grigori, E. G. Boman, S. Donfack, and T. A. Davis, *Hypergraph-based unsymmetric nested dissection ordering for sparse LU factorization*, SIAM J. Sci. Comput., 32 (2010), pp. 3426–3446.

[35] A. Gupta, *Fast and effective algorithms for graph partitioning and sparse matrix ordering*, Technical report RC 20453, IBM T. J. Watson Research Center, Yorktown Heights, New York, 1996.
[36] A. Gupta, *Watson graph partitioning package*, Technical report RC 20453, IBM T. J. Watson Research Center, Yorktown Heights, New York, 1996.
[37] B. Hendrickson and T. G. Kolda, *Graph partitioning models for parallel computing*, Parallel Comput., 26 (2000), pp. 1519–1534.
[38] B. Hendrickson and T. G. Kolda, *Partitioning rectangular and structurally unsymmetric sparse matrices for parallel processing*, SIAM J. Sci. Comput., 21 (2000), pp. 2048–2072.
[39] B. Hendrickson and R. Leland, *A multilevel algorithm for partitioning graphs*, in Proceedings of the Supercomputing '95, ACM, New York, 1995.
[40] B. Hendrickson and E. Rothberg, *Improving the run time and quality of nested dissection ordering*, SIAM J. Sci. Comput., 20 (1998), pp. 468–489.
[41] A. B. Kahng, *Fast hypergraph partition*, in Proceedings of the 26th ACM/IEEE Design Automation Conference, 1989, pp. 762–766.
[42] R. M. Karp, *Reducibility among combinatorial problems*, in Complexity of Computer Computations, R. E. Miller and J. W. Thatcher, eds., Plenum Press, New York, 1972, pp. 85–103.
[43] G. Karypis and V. Kumar, *A fast and high quality multilevel scheme for partitioning irregular graphs*, SIAM J. Sci. Comput., 20 (1998), pp. 359–392.
[44] G. Karypis and V. Kumar, *MeTiS A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices Version* 4.0, Department of Computer Science and Engineering, Army HPC Research Center, University of Minnesota, Minneapolis, MN, 1998.
[45] E. Kayaaslan, A. Pinar, U. V. Çatalyürek, and C. Aykanat, *Hypergraph partitioning through vertex separators on graphs*, Technical reports OSUBMI-TR-2010-n01 and BU-CE-1016, Department of Biomedical Informatics and Bilkent University, Computer Engineering Department, The Ohio State University, Columbus, OH, 2010; also available from http://bmi.osu.edu/hpc/publications.html.
[46] E. Kellerman, *Determination of keyword conflict*, IBM Technical Disclosure Bulletin, 1973.
[47] L. Kou, L. Stockmeyer, and C. K. Wong, *Covering edges by cliques with regard to keyword conflicts and intersection graphs*, Commun. ACM, 21 (1978), pp. 135–139.
[48] J. W. H. Liu, *Modification of the minimum degree algorithm by multiple elimination*, ACM Trans. Math. Software, 11 (1985), pp. 141–153.
[49] J. W. H. Liu, *The minimum degree ordering with constraints*, SIAM J. Sci. Stat. Comput., 10 (1989), pp. 1136–1145.
[50] D. J. Rose, *A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations*, Graph Theory and Computing, Academic Press, New York, 1972, pp. 183–217.
[51] E. Rothberg, *Ordering sparse matrices using approximate minimum local fill*, in Second SIAM Conference on Sparse Matrices, 1996.
[52] W. F. Tinney and J. W. Walker, *Direct solution of sparse network equations by optimally ordered triangular factorization*, in Proc. IEEE, 55, 1967, pp. 1801–1809.
[53] B. Uçar and C. Aykanat, *Revisiting hypergraph models for sparse matrix partitioning*, SIAM Rev., 49 (2007), pp. 595–603.
[54] M. Yannakakis, *Computing the minimum fill-in is NP-complete*, SIAM J. Algebraic Discrete Methods, 2 (1981), pp. 77–79.